INL/EXT-11-20798

Implementation of a New DTSTEP Algorithm for Use in RELAP5-3D and PVMEXEC Completion Report

George L. Mesina

December 2010

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-11-20798

Implementation of a New DTSTEP Algorithm for Use in RELAP5-3D and PVMEXEC Completion Report

George L. Mesina

December 2010

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Naval Reactors
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

INL/EXT-11-20798

Implementation of a New DTSTEP Algorithm for use in RELAP5-3D and PVMEXEC Completion Report

Dr. George L Mesina

December, 2010

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-11-20798

Implementation of a New DTSTEP Algorithm for use in RELAP5-3D and PVMEXEC Completion Report

Dr. George L Mesina

December, 2010

Idaho National Laboratory
Thermal Science and Safety Analysis
Idaho Falls, Idaho 83415
http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Naval Reactors
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

**Implementation of a New DTSTEP Algorithm
for use in RELAP5-3D and PVMEXEC
Completion Report**

INL/EXT-11-20798

December, 2010

Approved by:

Dr. George Mesina Author	Date
Robert Martin Technical Reviewer	Date
Hope Forsmann Technical Reviewer	Date
Dr. James Wolf Project Manager	Date
George Griffith Department Manager	Date

EXECUTIVE SUMMARY

The PVM Coupling methodology for decomposing a complex model into domains onto which individual programs may be applied has proven effective for solving many multi-physics problems. There have been, from the outset, some detailed and/or long-running models that cause the process to fail. Some 26 errors are listed in Tables 1 and 18 and in Section 11. These arise from deficiencies in the floating point calculation and testing of time steps, cumulative time, and time targets, along with unforeseen subtleties in the coupling technology.

This project addressed the PVM coupling issues surrounding the DTSTEP subroutines on RELAP5-3D and PVMEXEC. The algorithmic replacement of floating point control of these items with an integer based time-step method resolved these and related issues. This report documents the theory and implementation of the integer timestep methodology.

This report also provides a great deal of information about DTSTEP for code development and debugging. Alphabetized lists of all variables in both DTSTEP subroutines (RELAP5-3D and PVMEXEC) and related Fortran modules are organized into numerous tables according the subroutine or module in which it occurs. Many internal subprograms were also created in both DTSTEP subroutines and in some modules. Alphabetized lists of these subroutines and functions, with brief subprogram descriptions, are stored in other tables.

Checklist

Good	Originality
Good	Scientific Relevance
Excellent	Completeness
Acceptable	Acknowledgement
Good	Organization
Excellent	Clarity of writing
Checked	No mention of funding source
Checked	No naming of funding source staff except in references to open literature
Checked	No mention of alternative names for RELAP5-3D or PVMEXEC

CONTENTS

Figures

Tables

1.0 Purpose and Scope

The purpose of this report is to document the changes to the timestep advancement and selection scheme in RELAP5-3D and PVMEXEC that were successfully implemented.

The algorithm development and work to implement the new algorithm under the constraints of the project is reported. This document covers developments and testing through released version r3d244b, developmental version prB2441b.

2.0 Introduction

2.1 Background

The PVM coupling capability was incorporated into RELAP5-3D beginning in 1999. Since then, several User Problems (UP) have been reported, including the sample appearing in Table 1.

The most common failure is by "hanging the machine." This means that processes participating in the coupled calculation have arrived at coding that waits for a particular message to arrive, but those messages have not been and cannot be sent. Thus no further progress can be made on the calculation. Another type of coupling calculation error is the failure to perform edits such as printed output, restart output, and even plot output. Some coupling-related errors include restarts that either fail or differ from the same transient run without restarting. Another class of errors involve problem that stop before the end of transient and those that do not stop at all.

Table 1. Some User Problems associated with Coupling Calculations and DTSTEP

UP Number	Description
03009	6000 s transient doesn't stop. No plot, major, or restart from 3000 s onwards
03019	Marviken deck gets no major or restart edit at end of transient
03022	pvmedax.ii problem core dumps during transient
03024	VHTR calculation stops at 1st timecard end 28800 s rather than 2nd card's end time 345600 s
04011	A 180 s transient w/ Dt=0.00005 (3600000 advancements) runs past end time.
05001	A long-running calculation wrote no restart edits beyond a certain time.
06020	Code stopped writing major edits after 277200 s for a VHTR deck.
06034	RCCS pebble bed reactor calculation wrote no major edits at 10,050 or 45,000.
08015	A 3 loop system calculation the code hangs possibly due to velocity flip-flop.

In Table 1, the UP number has the form XXyyy, where XX are the last two digits of the year, and yyy is the number of the problem in the sequence they were submitted that year.

Not all of the UP listed in Table 1 were ever resolved. Many of those listed were solved by modifying the floating point calculations and/or tests. However, as examination of the descriptions will attest, the same kind of problem occurred later for another user input model.

In more recent years, code hangs were encountered with more detailed models and often with long-running transients. These are recorded in Sec. 10.1 Table 18.

Permanent resolution of all these errors was the purpose of the project described herein. These errors are related to the timestepping algorithms embodied in the subroutines named DTSTEP in both RELAP5-3D and PVMEXEC, but primarily in RELAP5-3D.

2.2 Original DTSTEP algorithm and issues

The creation of PVMEXEC, Sec. 19 Ref. (a), and the modification of RELAP5-3D, Sec. 19 Ref. (b), to operate with PVMEXEC required a fundamental change to the timestep selection and advancement scheme. Originally, the user-selected timestep controls, ΔT_{max} and ΔT_{min}, were the primary control mechanism. First, for all timesteps:

$$\Delta\, T_{max} \geq \Delta\, t_i \geq \Delta\, T_{min} \text{ for every timestep i} \tag{1}$$

Second, the algorithm enforces a halving/doubling timestep adjustment, subject to (1). The algorithm cuts the timestep in half and repeats the advancement from the previously successful time level when limits, such as mass-error upper bound or material Courant limit, are exceeded during the new advancement. Other conditions allow timestep doubling, but only after an even number of advancements with its current timestep. Therefore, with the exception of Rule (4) below, all timesteps have the form:

$$\Delta\, t_i = \Delta\, T_{max}/2^k, \text{ subject to (1)} \tag{2}$$

Third, the cumulative time $T_J = \sum_{i=1}^{i=J} \Delta\, t_i$ attained every integer multiple of ΔT_{max} up to the final time, regardless of the timesteps taken. A mathematical expression of this is: For every integer j, there is a J for which:

$$j \times \Delta\, T_{max} = T_J \; (\equiv \sum_{i=1}^{i=J} \Delta\, t_i) \tag{3}$$

Because floating point addition is inaccurate due to roundoff error, this rule was enforced by use of integer-based control. The singular exception to rules (2) and (3) was a timecard endtime, E, a user-selected time target that is not a multiple of ΔT_{max}. If a slightly larger timestep, up to 110% of ΔT_{max}, could reach the endtime, the necessary timestep was taken, even if it did not have form (1). Algorithmically,

$$if\ (T_i + 1.1\ \Delta\, t_{i+1} \geq E)\ then\ \ \Delta\, t_{i+1} = E - T_i \tag{4}$$

Because of rules (3) and (4), all RELAP5-3D user-requested output (major and minor edits, plots and restarts) is written only at multiples of ΔT_{max} or at timecard endtimes and there were no issues with output.

With the addition of coupling to RELAP5-3D, there is need to reach arbitrary communication points, C_M, that may violate rule (3) without being a timecard endtime. Resolution of this

requirement could be handled three ways: a new algorithm, a generalization of the existing algorithm, or restrictions on user input. The latter was deemed unacceptable as it would require code users to ensure that the same timestep input was used in the PVMEXEC and RELAP5-3D input decks. A choice was made to generalize the existing timestep algorithm.

Inaccuracies in floating point arithmetic make reaching unusual communication time targets difficult if PVMEXEC and RELAP5-3D are on different compute platforms. This is caused by the well-known issue of accumulation of floating point round-off error. Table 2 shows that starting at the same time and taking the same sized steps results in different cumulative times on a 64-bit floating-point Opteron vs. AMD 64 chip. Moreover, on a given processor, using a smaller timestep to arrive at the same target time (1005 sec) produces different roundoff error.

Table 2. Example of tolerance test "Type I Error" at a communication point

Specification	**T_{start}=1000.0, N = 100 timesteps**	**Δt = 0.005**
Opteron	T = 1005.0000000745	
AMD 64	T = 1004.999999888	
Specification	**T_{start}=1000.0, N = <u>1000</u> timesteps**	**Δt = <u>0.0005</u>**
AMD 64	T = 1005.000000238	

Since communication points, C_M, may not occur at multiples of ΔT_{MAX}, the integer-based control for reaching output points could not apply and it is clear that a floating point equality test would fail due to roundoff. A standard computer science solution is to apply a floating-point tolerance test rather than an equality test. In generalizing the algorithm, all communication and output times were evaluated using tolerance tests. The test function and test form were:

$$C(T_i,\ \Delta t) \equiv |T_i + \Delta t - C_M| \tag{5}$$

$$if\ (C(T_i,\ \Delta t_{i+1}) < \varepsilon)\ perform\ communication \tag{5a}$$

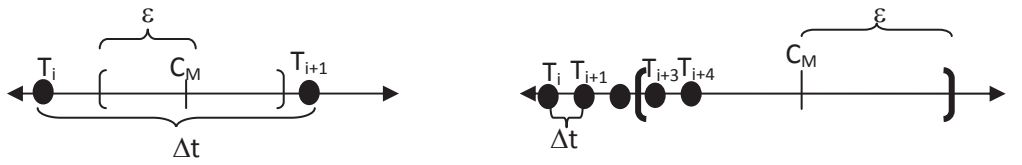
Communication between RELAP5-3D and the PVMEXEC took place when cumulative time was sufficiently close to communication target, C_M. The tolerance, ε, was taken to be larger than the round-off error produced in the test problems, but small enough to force communication at the correct time; $\varepsilon = 10^{-8}$ was used.

There were two opposing problems with the approach. The first problem occurs when round-off alters cumulative time by more than ε. In Table 2, round-off is in excess of $2x10^{-8} > \varepsilon$. Thus, a tolerance test of the form (5a) can activate at the wrong time or can be entirely bypassed. For example, with $T_{start} = 1.0$, $C_M = 1.001$, $\Delta T_{max} = 10^{-2}$, $\Delta T_{min} = 10^{-7}$, $\Delta t_{smallest} = 1.52587887214593x10^{-7}$ (via repeated halving). If every timestep were at this smallest Δt, the results are in Table 3.

Table 3. Example of tolerance test "Type I Error" at a communication point

Step #	T_i = Cumulative Time	$C(T_i, \Delta t) = \|T_i + \Delta t - C_M\|$
6553	1.0009990842492	$1.38x\ 10^{-7} > \varepsilon$
6554	1.0010000610128	$1.43x\ 10^{-8} > \varepsilon$

The difference is calculated according to Eq (5). Since both differences exceed ε, the test (5a) can never activate; as shown in Figure 1. Several User Problems, UP, were reported that had this at their source. In statistical terms, the null hypothesis is "test is true when T_i is closest." This is always rejected, even when true (on step 6554). Rejecting the null hypothesis when it is true is called Type I error or a false positive. *Type I error can be overcome by making ε larger*. For example, if $\varepsilon = 2x10^{-8}$, the test would activate on step 6554. It could also be overcome by making ΔT_{min} smaller; this leads to a problem with type II error as discussed next.



Type I Error: Bypass communication pt. *Type II Error: Activate before closest value*

Figure 1. Pictorial presentation of tolerance test Type I and II errors

In fact, RELAP5-3D input allows $\Delta t_{MIN} \geq 10^{-12}$; so $\Delta T_{min} < \varepsilon = 10^{-8}$ is allowed. With such input, tolerance tests like (5a) will activate on the very first time $T_i + \Delta t_{i+1}$ is within $\varepsilon = 10^{-8}$ of C_M. As shown in Figure 1, this occurs *before* the *intended activation time* (the closest cumulative time). The test activates when it should not. Accepting the null hypothesis when it is false is called Type II error or a false negative. *Type II error can be overcome by making ε smaller*.

It is clear that *altering ε can never overcome both types of error* as Type I error requires ε to be increased while Type II error requires ε to be reduced. Nevertheless, attempts were made to determine a value of ε that was large enough such that round-off error could be mitigated and small enough to preclude false negatives. Application of the coupled code system to complex and long running problems of interest revealed several failures in the floating point-based algorithm. Other slight modifications of the floating point timestep algorithm were introduced; failures were resolved, however, the general Type I and Type II error problems were not.

3.0 Integer Timestep Concept

Since the floating point tolerance test was insufficient to resolve the combined Type I and Type II error issue that was caused by floating point round-off, the INL proposed to develop an integer based algorithm for time-stepping to replace it. The fundamental reason was that integer addition, subtraction and multiplication are exact; there is no round-off. However, *as PVMEXEC and other all codes connecting with RELAP5-3D use real time for output, communication and timecard end, the integer algorithm must also use floating point values for these*.

Use of integers to count time is not a new idea. Though time is inherently a continuous quantity that is best represented by floating point values on a computer, integer-based time is used in all computers. Computer chip speed is rated in terms of the number of billion operations it can

perform in a second in GHz or clock-cycles. A clock-cycle is the amount of time to perform the simplest operation or *one* tick of its clock. The integer basis for time stepping is also the clock-cycle or tick and that is defined to be the smallest timestep that RELAP5-3D can take and is denoted Δt_{small}. It is generated by the user's selection of ΔT_{max} and ΔT_{min}. An example is shown in Figure 2 with real time denoted t, integer time denoted τ, Δt_{max} = 0.004, and Δt_{min} = 1.0e-7.

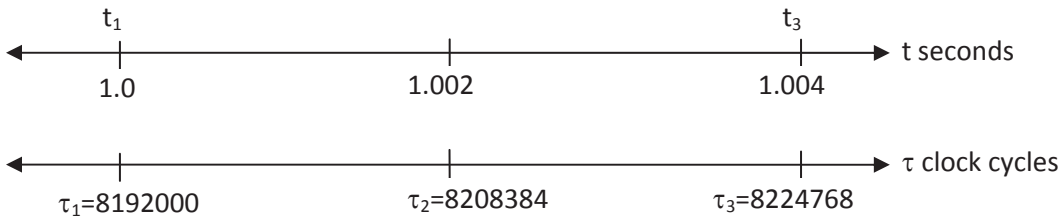


Figure 2. Correspondence between integer and real time

In this case, the integer clock cycle value τ=1 tick, represents real time Δt_{small}=1.220703125x10^{-7} and $\Delta\tau_{max} = 2^{15}$ because 2^{15}x(1.22x10^{-7}) = 0.004. Thus, t_1 = 1.000 corresponds to $\tau_1 = 250x2^{15}$ = 8192000, and $\tau_2 = \tau_1 + 2^{14}$ = 8208384 because .002 = 2^{14}x(1.22x10^{-7}).

The formulae that generate the correspondence between reals and integers are given New Algorithm 1 which encompasses Eq. (6), (6a), and (6b).

New Algorithm 1. Integer/Real Time Initialization (start of transient)

1. $H = Floor\left[log_2\left(\frac{\Delta T_{max,C}}{\Delta T_{min,C}}\right)\right] = Floor[log(\frac{\Delta T_{max}}{\Delta T_{min}})/log(2)]$ (6)
2. $\Delta\tau_{max} = 2^H$ (6a)
3. $\Delta t_{small} = \Delta T_{max}/\Delta\tau_{max} = \Delta T_{max}/2^H$ (6b)

Note: the floor function in Eq. (6) truncates real numbers by removing the fractional portion. In Figure 2, log(.004/.0000001)/log(2) ≈ 15.28; so H = Floor[15.28] = 15, $\Delta\tau_{max} = 2^{15}$ = 32768, and Δt_{small} = .004/32768 = 0.0000001220703125.

There is no concern about restricting the range of floating point values that can be represented by 64-bit integers. A 64-bit integer can represent values in excess of 9x10^{18}. This is sufficient, for example, to exactly calculate every integer multiple of Δt_{small}, between $\Delta T_{MIN} = 10^{-9}$ and T=10^9 seconds.

Integer timestepping has replaced real timestepping in RELAP5-3D and PVMEXEC. However, most of the complication occurs in RELAP5-3D; therefore most of the ensuing discussion is presented from the perspective of the RELAP5-3D code. Reference is made to PVMEXEC where helpful and in the PVMEXEC section.

4.0 Integer Timestep Basics

This Δt_{small} is the smallest value of Δt that satisfies rules (1) and (2), but it can change on every timecard, C, because the user can change the max and min timestep, $\Delta T_{max,C}$ and $\Delta T_{min,C}$. The calculations for crossing to timecard C-1 to timecard C are given in New Algorithm 2.

New Algorithm 2. Timecard crossing

1. $H(C) = Floor[log(\frac{\Delta T_{max,C}}{\Delta T_{min,C}})/log(2)]$
2. $\Delta\tau_{max,C} = 2^{H(C)}$
3. $\Delta t_{small,C} = \Delta T_{max,C}/\Delta\tau_{max,C}$
4. $\tau_C = 0$

Step 4 is the only real difference from New Algorithm 1. It restarts integer cumulative time at zero with every timecard while floating-point cumulative time is unaffected. Throughout the time controlled by timecard C, integer cumulative time, floating point Δt, and floating point cumulative time are calculated by the Equations of New Algorithm 3. Note that T_{C-1} is the endtime of the timecard C-1; it is also the beginning time of timecard C.

New Algorithm 3. Integer/Real time advancement (for chosen $\Delta\tau_i$)

1. $\tau_{C,i} = \tau_{C,i} + \Delta\tau_i$ (7)
2. $\Delta t_i = \Delta t_{small,C} * \Delta\tau_i$ (7a)
3. $T_i = T_{C-1} + \tau_{C,i} * \Delta t_{small}$ (7b)

Floating point cumulative time and time targets sent at a communication points by PVMEXEC to RELAP5-3D *must be converted to integers*. Time targets are communication, output, or timecard end times. Floating point time targets may not be representable exactly by an integer multiple of $\Delta t_{small,C}$. An unusual time target is not a multiple of $\Delta t_{small,C}$, a normal time is.

An unusual time can be created by user selection of a timecard endtime that is not a multiple of ΔT_{max}. It can also be caused by PVMEXEC having a different ΔT_{max} than RELAP5-3D. Time targets, T_{Targ}, created by PVMEXEC are integer multiples of its Δt_{small}, namely $\Delta t_{small,\,C',\,P}$. Time targets are unusual if they are not multiples of the Δt_{small} of RELAP5-3D, namely $\Delta t_{small,\,C,\,R}$.

If T_{Targ} is unusual, there is no exact integer representation for it. The corresponding value on the τ-timeline lies between two consecutive integers. Figure 2a has $T_{Targ} = 1.0021$ and $T_{C-1} = 1.0$; thus $\tau_{left} = [(T_{Targ} - T_{c-1})/\Delta t_{small}] = 8192000 + 17203$ (truncated) and $\tau_{right} = \tau_{left} + 1 = 8209204$.

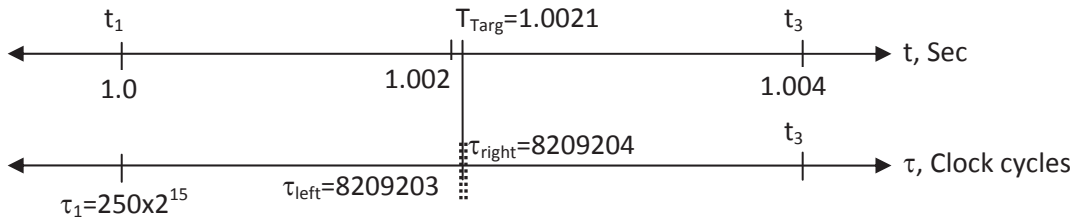


Figure 2a. Unusual times cannot be represented exactly by an integer.

Of course, if T_{Targ} is a normal time, then integer τ_{Targ} = Round[$(T - T_{C-1})/\Delta t_{small}$] exactly represents the real time target and there are no issues reaching it.

Let $T_{E,k}$ be the next minor edit, plot, major edit, restart, and PVM-communication points from the PVM message, respectively, and let $\tau_{E,k}$ be the integer equivalents. The conversion of floating point time quantities to integers is given in New Algorithm 4. Rounding is necessary to prevent automatic truncation of fractional portions of real quotients in integer conversion. For example, applying Eq. (8) to Table 3 with T = 1.0009990842492, Δt_{small} = 1.52587887214593x10^{-7} and T_C = 1.0, yields τ_C = 6547 without rounding, but 6548 with rounding.

New Algorithm 4. Convert real time quantities to integers

1. $\tau_C = f(T, T_{C-1}, \Delta t_{small}) \equiv$ Round$[(T - T_{C-1})/\Delta t_{small}]$ (8)
2. $\Delta\tau_i$ = Round $[\Delta t_i/\Delta t_{small}]$ (8a)
3. $\tau_{E,k}$ = Round$[(T_{E,k} - T_{C-1})/\Delta t_{small}]$, j=1, . . . ,5 (8b)

This algorithm applies to unusual or normal time quantities. However for unusual quantities, the integer does not exactly represent the floating point quantity.

Another important concept is the so-called *perfection of the timestep, ∆t*. This means making a normal timestep into an exact multiple of $\Delta\tau_{small}$. It is necessary since floating point calculations of Δt elsewhere in the overall DTSTEP algorithm may result in a Δt_i that is not an integer multiple of $\Delta\tau_{small}$. Just before advancing time, a normal timestep is perfected with New Algorithm 5.

New Algorithm 5. Perfect a normal timestep

1. Normal = $(|\tau_{i+1} * \Delta t_{small} - \Delta t_{i+1}| < \delta)$ (9)
   If (Normal) then
   a. $\Delta t_{i+1} = \Delta t_{small,C} x \Delta\tau_{i+1}$ (7a repeated)
   b. $T_i = T_{C-1} + (\tau_{C,i}) x (\Delta t_{small})$ (7b repeated)

If the timestep is close to a multiple of $\Delta\tau_{small}$, then inexactness in its last bits resulting from roundoff in its calculation is eliminated by (7a). The same is done for cumulative time. Since Δt_{i+1} does not arise from a sum of many terms, accumulation of roundoff is not an issue; thus roundoff is confined to the last one or two bits, depending upon the calculation. Therefore, δ is taken to be a very tight tolerance. Initially, $\delta = 10^{-10}$ was used; however, for very large timesteps, such as in the ans79.i input model with timesteps on the order of 10^6, the tolerance must scale with the size of the timestep, and therefore $\delta = \Delta t_{i+1} x 10^{-10}$ was used.

These conversions provide a basis for replacement of floating point tolerance tests with integer equality tests for reaching a time target. This issue is explained in Section 4.1.

4.1 Integer Time Equality Tests

If T_{Targ} is normal, then integer τ_{Targ}, defined via Eq. (8b), exactly represents the real time target and there are no issues reaching it. For unusual time targets there is considerable complication.

The idea for handling unusual times is to consider two linked timesteps that will reach from the current advancement, i, at time T_i to advancement i+2 at $T_{i+2} = T_i + 2\Delta t_i$. During the first linked step, integer and real time do not correspond exactly. The timestep used by RELAP5-3D during advancement i+1 is the value of Δt that exactly reaches T_{TARG}. The second step reaches T_{i+2}.

New Algorithm 6: Unusual Time Target Handler

If (UH is false) UH = ($T_i + \Delta t_{i+1} \geq T_{TARG}$) and ($T_{TARG}$ is unusual)
Linked Advancement 1 when UH is true

1. $\Delta t_{i+1} = T_{TARG} - T_i$
2. $\Delta \tau_{i+1} = \tau_{left} - \tau_i$
3. $\tau_{save} = \tau_i + 2\Delta\tau_i$
4. $T_{save} = T_i + 2\Delta t_i$

Linked Advancement 2, when UH is true,

1. $\Delta t_{i+2} = T_{save} - T_{TARG}$
2. $\Delta \tau_{i+2} = \tau_{save} - \tau_{i+1}$
3. UH = false

Variable UH indicates that the unusual handler is active when true, otherwise it is false. Once the handler is on, UH keeps it on until the second linked advancement is finished. There are two complications with New Algorithm 6. The first is determining that T_{TARG} is unusual; see Section 4.1.1. The second occurs when the second linked step is halved by other portions of the DTSTEP algorithm; see 4.1.2. Also, this algorithm does not cover the 110% stretch logic of rule (4).

4.1.1 Normal / Unusual Testing

Algorithm 6 applies to unusual targets, those that are not exactly a multiple of Δt_{small}. However, it is important that times very close to multiples of Δt_{small} be considered normal also. Otherwise accumulation of any roundoff error would cause virtually every time value produced via floating point summing to test as unusual. Another consideration besides roundoff is the variation in the quotients when divided by Δt_{small}. Quotients of larger numbers, such as T, can have less accuracy than those of smaller numbers, such as Δt. Therefore, determination of unusual time target is better served by basing the test for normality on Δt rather than T_{TARG}.

There are two coupling modes based on control of the timestep by PVMEXEC. In asynchronous coupling modes, all codes proceed to a communication point with their own selection of Δt at every step. In synchronous mode, PVMEXEC supplies Δt_{i+1} to all coupled codes at every step, and all codes must use that timestep. This is summarized in Eq. (9).

$$\Delta t_{i+1} = \begin{Bmatrix} T_{TARG} - T_i & if\ asynchronous \\ \Delta t_{PVMEXEC,i+1} & if\ synchronous \end{Bmatrix} \quad (9)$$

With the timestep selected, the determination of unusualness is made with New Algorithm 7.

New Algorithm 7. Time Target Unusualness Test

1. $\Delta\tau_{check}$ = Round $[\Delta t_{i+1}/\Delta t_{small}]$ (10)
2. $\Delta t_{ch} = \Delta\tau_{check} * \Delta t_{small}$ (10a)
3. UNUSUAL = $|\Delta t_{ch} - \Delta t_{i+1}| > 0.01 x \Delta t_{small}$ (10b)

Algorithm 7 sets UNUSUAL to true when the target is an unusual time. Since Δt_{i+1} and Δt_{ch} have the same size and agree for many leading digits. The number of leading digits that must agree determines unusualness. Requiring too many allows accumulation of roundoff to falsely indicate unusual status when the target is normal. Not requiring enough allows some unusual targets to falsely test as normal. This is a Type I vs. Type II error situation.

Note that for smaller timesteps than Δt_{max}, the subtraction in Eq. (9) reduces the number of significant digits available to compare. The test reflects this loss. Comparison against $0.01 x \Delta t_{small}$ in Eq. (10b) requires at least two decimal places of agreement when Δt_{i+1} is close to Δt_{small} in size, and requires more places of agreement for larger Δt_{i+1}. Typically, $\Delta T_{max}/\Delta T_{min} \approx 10^6$, so typically, agreement to 6+2=8 places is required for Δt_{i+1} near ΔT_{max}. For IEEE 64-bit floating pt. representation, approximately 15 digits can be represented fully. Therefore, agreement to the first 8 places ignores the lowest 7=15-8 respectively.

Since New Algorithm 5 perfects every normal timestep and its associated cumulative time, T_i, only the final bit of T_i is subject to roundoff error. However for a large cumulative time, the difference $\Delta t_{i+1} = T_{Targ} - T_i$ in Equation (9) can account for a substantial loss of significant digits, creating an inaccuracy in the lowest 7 decimal places. This problem is itself mitigated on most modern platforms that use 96 bits for floating point operations (effectively adding 10 extra digits to the calculation) before rounding to store the value. Thus it is very difficult, but possible if all 15 digits of time are important, for New Algorithm 7 to fail.

It is possible to improve New Algorithm 7 so that failure is even less likely. By combining Eq. (6b), (2) and (7a) it is seen that a normal <u>integer</u> timestep is always a power of two.

$\Delta t_{small} = \Delta T_{max}/2^H$, so that $\Delta T_{max} = 2^H \Delta t_{small}$
$\Delta t_i = \Delta T_{max}/2^k = \Delta t_{small} 2^H/2^k = \Delta t_{small} 2^{H-k}$
$\Delta\tau_i = \Delta t_i/\Delta t_{small} = 2^{H-k}$.

Any timestep taken to reach a time target whose integer representative is not a power of two is automatically unusual. A simple test for this is given in New Algorithm 8.

New Algorithm 8. Power of two test for normality

1. H = Round[log($\Delta\tau_{i+1}$)/log(2)]
2. NORMAL = ($\Delta\tau_{i+1}$ .eq. 2^H)

The combination of Algorithms 7 and 8 provides a better test for normality or unusualness than Algorithm 7 alone.

4.1.2 The Unusual Handler's second linked step

The second linked step of Algorithm 6, Unusual Time Target Handler, must handle two cases:
(1) Δt is halved by other portions of the DTSTEP algorithm or
(2) The advancement is repeated.

Case (2), repeating either the first or second advancement causes a special logical variable to be set to indicate that time backed up by one timestep. On the next pass through DTSTEP, the unusual handler is cancelled.

Case (1), if the first linked step succeeds and the second is halved and repeated, the saved target T_{i+2} cannot be reached on the second step. However, the handler is cancelled by the backup and the rest of the DTSTEP algorithm will create an appropriate timestep.

4.2 Integer equality tests

With these definitions, conversion equations, and algorithms in place, all the floating point tolerance tests were replaced with integer equality tests.

Replace statements with this form,

$$\text{If } (\text{abs}(T_i + \Delta t_i - T_{TARG}) < \varepsilon) \text{ action}$$

With

$$\text{If } (\tau_i + \Delta\tau_i \text{ .eq. } \tau_{TARG}) \text{ action} \qquad (11)$$

There were also tests that required integer inequalities, such as stretch logic. There were a couple of tests that had to remain floating point, such as a Courant limit test. Converting all tests proved insufficient to solve all the problems with the PVM coupling of RELAP5-3D to other codes. Some of these are covered in Section 5.

5.0 Integer Time Advanced Concepts

Advanced concepts covers the solution to difficult programming issues due either to complexity in the algorithm underlying the original coding, or to the need to create alternative means to implement with integers an algorithm designed for floating point calculations. The latter were often revealed by runtime errors. Most errors with the integer time step involved integer cumulative time differing from a time target by exactly one. Most of these involved time targets generated by PVMEXEC and sent to RELAP5-3D. Under close examination, the source of each of these was tracked and solutions devised.

Before examining these issues, it is important to note two conditions that were placed upon the workscope as they restricted the choice of solutions. The development and implementation of the integer algorithm was carried out under two constraints:

1. *The functionality of DTSTEP must remain the same*.
2. *The PVM messages cannot be altered in any way, neither by adding new communication messages nor by modifying any existing message*.

The first constraint was put in place to ensure that the tens of thousands of existing input models would continue operating exactly the same before and after the change to the algorithm, except that problems which failed under the tolerance test were to run under the integer algorithm. It was recognized cumulative time would be slightly different because of more accurate temporal calculation; however this is not a change of functionality.

The second constraint was put in place to preclude the necessity of modifying all the computer codes that had previously been programmed to communicate with RELAP5-3D and PVMEXEC. Although efficient at reducing overall rework, two exceptions to this constraint proved necessary for solving difficult issues; these were eventually implemented.

In the interest of brevity, only five of the many and complex advanced issues are covered, and only from a high level perspective. Those addressed include: handling minimal timesteps in Sec. 5.1, finding a target halving of ΔT_{max} in Sec 5.2, timecard crossing in Sec 5.3, message exchange time in Sec 5.4, the 110% stretch logic in Sec 5.5, and the simplification of the synchronous coupling handler in Sec. 5.6.

5.1 Handling Minimal Timestep

The DTSTEP algorithm originally automatically halved a timestep if certain conditions occurred, such as excessive mass error, even if already minimal, temporarily making $\Delta t_{i+1} = \Delta t_{small}/2$. Later, the algorithm would correct this by doubling Δt_{i+1} if below ΔT_{min}. The reduction below ΔT_{min} occurred in several places in DTSTEP. This algorithm does not work for integers because the smallest integer is 1; halving 1 produces 0, and doubling 0 produces 0, not 1.

The solution was to *rework the DTSTEP algorithm* so that timesteps were not allowed, under any circumstances, to go below 1 for integers and for ΔT_{min} for reals, and *automatic doubling was adjusted* to account for this. In compliance with the requirement that DTSTEP functionality be unchanged, it is noted that this had no effect on the functionality of DTSTEP. See Section 5.0.

5.2 Exceptions to the halving and doubling logic for ΔT_{max}

There are several instances in the overall DTSTEP algorithm when the current Δt does not fit the DTSTEP halving and doubling scheme, rule (2). These instances are:

1. When crossing to a timecard with a different ΔT_{max}, different ΔT_{min}, or both
2. When ΔT is calculated to satisfy Courant limit
3. After a 110% stretch to a time target has occurred
4. After a compression to take 2 equal steps to reach a time target

In all these cases, the time step in use does not match the form of rule (2). Return to the correct form is required so that the halving/doubling logic of DTSTEP works properly. This is done in accordance with Section 5.0 requirement that the functionality of DTSTEP not be changed.

In the original DTSTEP algorithm, a timestep halving loop starting at the controlling value of Δt_{max} and halving the candidate timestep was used. It stopped with the first value less than or equal to the target value. It could produce a value below Δt_{small}.

This is unworkable for the integer algorithm as explained in Section 5.1. New Algorithm 9 was created that stopped if $\Delta t_{candidate} < \Delta t_{small}$, and set $\Delta t_{candidate} = \Delta t_{small}$ and set $\Delta\tau_{candidate}$ using Eq. (8a). If the target was negative, it set the failure flag to true so that the RELAP5-3D calculation could terminate gracefully.

New Algorithm 9. Find Halving ≤ target

If (target < 0.0) then
    Fail = true
Else if (target < Δt_{small}) then
    $\Delta t = \Delta t_{small}$
Else
    Loop (on k) to find $\Delta t = (0.5)^k \Delta T_{BASE} \leq$ target
    where ΔT_{BASE} is either ΔT_{MAX} or an unusual time-step
Endif
$\Delta\tau$ = Round[$\Delta t/\Delta t_{small}$]

Note that the halving is actually performed by multiplication by the reduction factor of 0.5. The test for negative target is included because during the debugging, negative timesteps were created in other portions of DTSTEP and this was introduced to catch those and stop gracefully.

5.3 Timecard crossing and endtime

From the original DTSTEP algorithm, a timecard crossing is governed by the logical variable LAST. This variable was set differently for PVM synchronous control, for which it could only be true at the transient end, than for all other modes of operation where it is true at timecard endtimes.

Under PVM asynchronous control or autonomous RELAP5-3D control, LAST became true only when, before a timestep can be taken, cumulative time is already past the timecard endtime. The integer equivalent test was an inequality, rather than an equality such as Eq. (11).

Under synchronous control, this test was restricted to transient endtime as calculated from the values of ΔT_{max} and ΔT_{min} and T_{end} on the most recent PVM communication of these values. Synchronous control was maintained according to Requirement 2 of Sec 5.0.

5.4 110% stretch logic

The stretch logic was applied to reaching every unusual time target. For PVM coupling, DTSTEP included stretch logic for plot, minor edit, major edit, restart, communication time and timecard endtime. There were two issues to resolve, conversion to integers and the proper order.

This test required a conversion to integers for the testing, but use of floating point time to calculate the potentially unusual timestep. In some cases, the 110% stretch logic actually caused a timestep to be shortened to an unusual step to exactly reach the next target time. Finally, the algorithm for processing the stretch differed between autonomous control, PVM synchronous and PVM explicit time control in RELAP5-3D.

With explicit time control in use, the explicit message exchange time was the most important time to reach. It was necessary to override the 110% stretch when taking a step of that size would step past an explicit message exchange time to reach a larger time target; that would result in a "code hang" or "hung machine." Both mean that one code is waiting for one kind of message while the other code waited for another, and no message is being sent by either. This created in an interminable period of waiting, otherwise called a "code hang," and required the coupled calculation to be terminated by outside intervention.

This induced a requirement to check and insure that explicit exchange time could never be bypassed. The functionality was preserved according to requirement 2 of Sec 5.0, but the algorithm was adjusted to account for this and solutions to issues identified during debugging. Subroutine dthyCalc (see Table 7b) embodies this algorithm; so it is not given here.

5.5 Message and Target Time

The time to exchange messages was a major source of programming issues for PVM coupling between RELAP5-3D DTSTEP and other codes, including PVMEXEC DTSTEP. Roundoff error often caused the codes to meet the exchange time condition on a different timestep resulting in a hung machine. This was an issue for both synchronous and asynchronous coupling.

Depending on whether coupling is synchronous, asynchronous, or standalone, different logic paths and calculations are performed and the messages are different or non-existent. The programming was complicated by insufficient data for the original RELAP5-3D DTSTEP algorithm that required end of timecard. No successful substitute for that datum was found; so eventually it had to be added despite Section 5.0 rule 2. Integer calculations were further complicated by exchange times that were unusual times.

To reduce rounding errors when converting floating point times from messages to integers, message times were recalculated, based on the controlling ΔT_{max} and ΔT_{min} at every message exchange. In fact for the same reason, whenever one target time was reached, every target time was recalculated.

5.6 Synchronous Mode Simplification

In the original implementation of the integer-based timestep size selection and advancement algorithm, there were situations in which RELAP5-3D, operating in a synchronous mode where the ultimate timestep size is determined by PVMEXEC, was performing calculations of the timestep size needed to reach special communication intervals. The logic that was used for some of the calculations was inconsistent between RELAP5-3D and PVMEXEC, but more importantly, the logic was completely superfluous. However, according to requirement 2 of Section 5.0, this superfluous logic and coding could not be eliminated.

After numerous efforts to correct the inconsistencies between RELAP5-3D and PVMEXEC, it was decided to make another exception to Sec. 5.0 requirement 2. This resulted in essentially a new algorithm for handling synchronous coupling. In the new algorithm, RELAP5-3D reports a timestep size back to PVMEXEC that depends only on the built-in error-checking and error-mitigation processes in the code, including the prohibition of violating the Courant Limit for

RELAP5-3D problems that use the semi-implicit time advancement scheme. This represents a significant reduction in the logic associated with time step size selection for synchronously coupled calculations.

6.0 Integer Time Implementation

Initial status is in Sec 6.1. The implementation had many parts: mapping DTSTEP functionality in Sec 6.2; moving code into collections according to functionality in Sec 6.3; defining, eliminating & mnemonically renaming variables in Sec 6.4; creating a module for integer data in Sec 6.5; and creating internal subroutines in Sec 6.6.

6.1 Initial Status

The original status of RELAP5-3D DTSTEP was poor. There was little documentation, numerous GOTO statements (both forward and backward) which made tracing the logic flow extremely difficult, non-mnemonic variable names, large amounts of inert or unused code marked with pre-compiler directives, even some sections of dead code. Over a dozen authors had written code and modified it; thus many programming styles were present. All this made the main algorithm hard to read and understand.
The PVMEXEC DTSTEP subroutine was created from the RELAP5-3D DTSTEP subroutine by removing coding unneeded for controlling the coupled calculation and then adding the communication and decision logic to operate as the executive. It had many of the same issues.

There were a number of existing and unresolved user problems before this project began (all of which have since been resolved); Table 1 lists several and indicates that debugging these two subroutines had been ongoing for 5 years. Moreover, there were also a number of undiscovered errors that this project revealed and resolved. Together, the pair of subroutines had algorithmic and coding issues as well as a number of errors.

6.2 DTSTEP Functionality

Mapping the DTSTEP functionality was the necessary starting point. It supported requirement one of Section 5.0 and proved essential to all ensuing work. The following is a high-level summary of the functionality within RELAP5-3D DTSTEP after removing all coding that is either inert or otherwise unused.

A. AUTOMATIC FUNCTIONS

1. Initialization
   - Unconditional initialization
   - Initialization carried out on the first transient step only
   - Card-one initialization
2. Advancement statistics collection

B. END OF ATTEMPTED-ADVANCEMENT PROCESSING

3. Time step success evaluation
   - Based on RELAP5-3D criteria, set indicator of step success, failure or backup
   - If synchronous hydraulic coupling, (A) send indicator to PVM executive, (B) Receive global success flag from PVM executive

- For failure – restore old time step, time and data, activate debug
- For backup - reduce time step and time, restore old data

4. Transient/steady-state termination conditions
   - RELAP5-3D termination conditions
     - Normal - Transient terminated by end of time step cards
     - Transient terminated by trip
     - Transient terminated by approach to CPU time limit.
     - Transient has reached steady state.
     - Transient terminated by interactive command.
     - Transient terminated by Card 105 input option.
     - Transient terminated by variable volume model.
   - Transient terminated by failure in PVM coupled computation
   - Help operations
5. Time targeting
   - *Current & next edit time– Based on time-step card and PVM target time data
   - *Exact arrival at time target – New integer-based time-target algorithm

C. SET-UP / PREPARATION FOR "NEXT" ATTEMPTED-ADVANCEMENT

6. PVM communication

NOTE: this is only done if success is indicated and it is not the last advancement.

- If at explicit asynchronous hydraulic coupling communication time, send/receive appropriate messages
- If explicit synchronous hydraulic coupling, send/receive appropriate messages
- If at a PVM (edit or explicit coupling time) target time, receive new set of PVM target times

7. RELAP5-3D time step calculation

Note: only the <u>final selection</u> sub-function is performed if success flag indicates failure.

- *Time step card control processing –control from current card, next card?
- *Courant limit calculation(s)
- *Time step halving and doubling – RELAP5-3D conditions that change time-step size
- Hitting the target – stretch above max, shrink below min, two equal steps
- <u>Final selection</u> of time step – Based on success/failure control, normal step selection, and hitting target. Update cumulative time and advancement count.

8. PVM synchronous time step communication
   - Send synchronous message – Transmit time-step to PVM executive
   - Receive synchronous message – Get global time-step

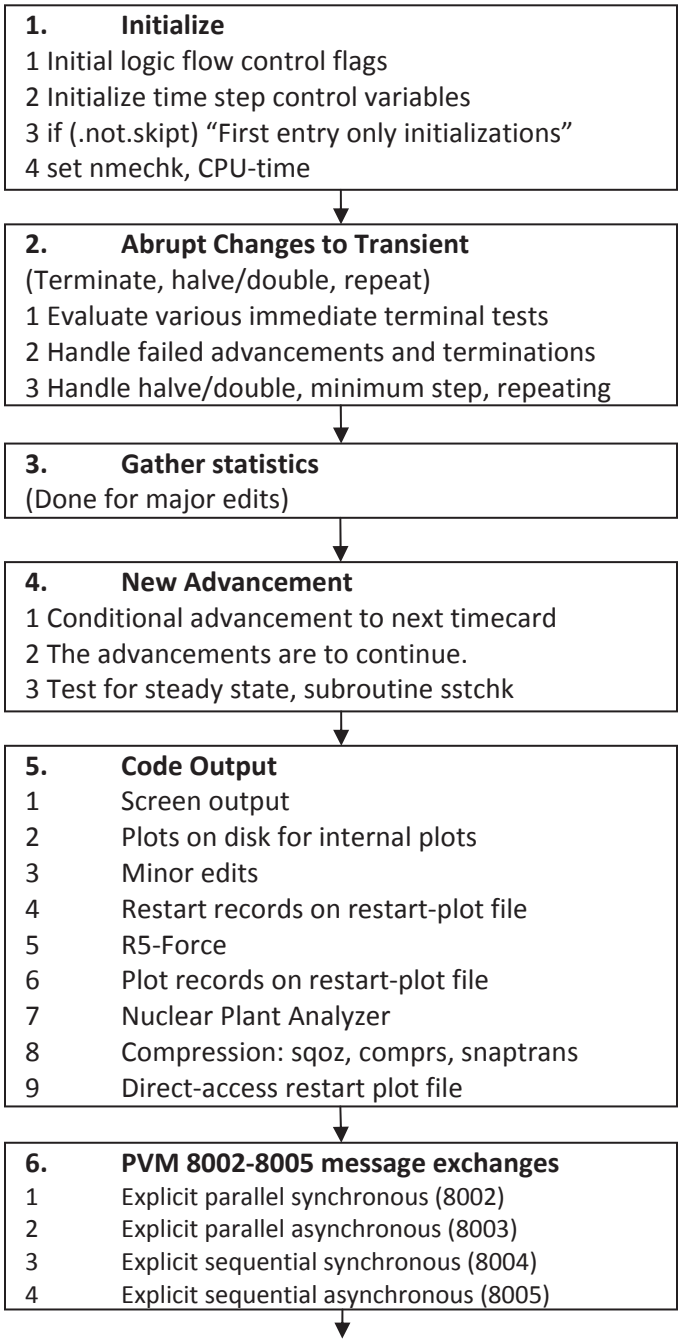
D. OUTPUT

9. Code output
   - Screen text
   - File output
     - Printed output file (major and minor edits)
     - Restart-plot files (plot records and restart records, sequential & direct access)
     - Dump files (binary files controlled by 105-card input)
     - Additional debug output (controlled by failure modes, 105-card input)

Note that the four major sections and nine subsections are arbitrary and introduced by the author merely to enhance understanding. The coding showed no such breakouts, nor did the coding for all these sections exist in contiguous blocks.

6.3 DTSTEP Reorganization

These functions identified in Sec 6.2 were reorganized into 11 major sections and coding was moved to collect the statements that worked together to perform these functions. This was done without affecting the logic flow. Figures 3a and 3b give a “sectional flowchart” of the reorganization of DTSTEP.



NOTE
This sectional flow diagram shows no arrows for jumps between or within sections.

Figure 3a. First portion of "Sectional Flowchart" for RELAP5-3D DTSTEP Functions

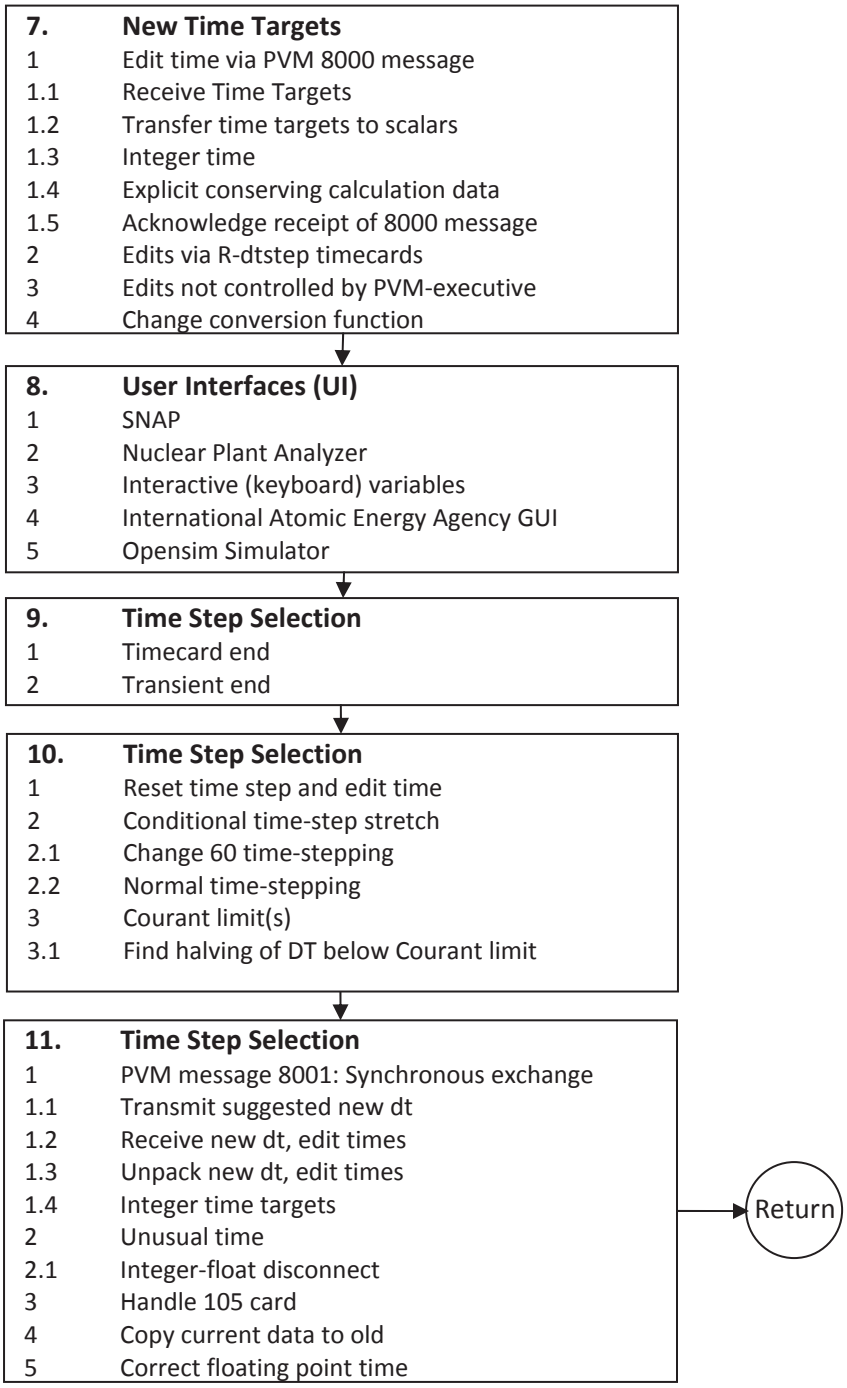


Figure 3b. Second portion of "Sectional Flowchart" for RELAP5-3D DTSTEP Functions

This sectional flowchart, when compared with the outline of DTSTEP in Section 6.1, shows code movement to recollect and unify functionality. It does not show jumps between sections. Figure 4 does.

Figure 4. DTSTEP High-level Flowchart showing jumps between sections.

In Figure 4, black is for normal flow and dark red jumps go to output. Other jumps shown lead to the subroutine exit. Jumps not shown are internal to a given flowchart section. In the initial DTSTEP coding, these included jumps around coding that did not belong to the section.

6.4 Define, mnemonically rename, eliminate variables, and fix declarations, create integer timestep variables

Before the conversion to integer timestepping could be undertaken effectively, workings of the DTSTEP subroutine had to be understood. Part of this was mapping and reorganizing the coding, but another part was deciphering the meaning of the variables and giving them more mnemonic names. This was undertaken simultaneously with the work of Sec 6.3.

All variables in DTSTEP *relevant to the project* were deciphered and documented in Table 4. The meaning of variables such as *gt19*, *iosini, nwqa,* and *j* (not a loop counter) then becomes clear for logic flow tracing, debugging, and conversion to integer timestepping.

Table 4. Data dictionary of all variables originally in DTSTEP

Key:	**Local variable names in lower case** **GLOBAL variable names in UPPER case**
Name	**Definition / Description**
af	time step Amplification Factor for time step (normally = 2.0).

aflag	(G) true in dtstep when heat structures are advance checked in htadv
CPUREM	CPU time REMaining - 5 elements from input file 105-line.
(1)	Terminate transient immediately whenever CPUREM(1) > time remaining on CPU clock for the run.
(2)	Terminate transient after successful time step whenever CPUREM(2) > time remaining on CPU clock for the run.
(3)	CPU time remaining for the run. On input, it is the time allocated for the run. Zero means no time limit.
(4)	> 0 Turn on debug on this attempted advancement. = -1 Write a dumpfile after advancement CPUREM(5) & quit. = -2 Same as -1, except don't quit, make 2nd advancement and write a second dumpfile (for comparison) & quit.
(5)	> 0 Terminate code on this advancement.
	= 0 Write dump file BEFORE starting transient & terminate.
curtmi	CURrent Time for next MInor edit.
curtmj	CURrent Time for next MaJor edit.
curtpl	CURrent Time for next PLot edit (time card option d = 4
curtrs	CURrent Time for next ReStart edit.
curtex	CURrent Time for next EXplicit coupling data exchange.
dtadj	the adjusted DT (doubled/halved).
dtctr	ConTRol DT.
dthyx	minimum of new adjusted HYdraulic DT and last time step value of dtX.
dtintv	DT INTerVal for minor/plot, major, or restart edits.
dtmax_i	DTMAX from I-th timecard.
dtmin_i	DTMIN from I-th timecard.
dtrem	time REMaining on cpu clock for this problem.
dtx	courant DT.
gt14	Go To statement label 14. "Repeat time step with old values for dump file" flag. False initially. Set true if cpurei(4)==-2
gt19	Go To statement label 19. set true if succes==2, false if dt<=dtmin_i
gt27	Go To statement label 27. true if cpurem(1)/=0 & dtrem<=cpurem(1) or succes==6 or help<0 & /=-2. Set false initially & when doing output
gt38	Go To statement label 38. help diagnostic step indicator. set true if help<=2.
gt39	Go To statement label 39. set false initially & when doing output set true if done==-7.
gt39a	Go To statement label 39 Also. Move (maybe) & Exit Flag. set true if cpurei(4)>0 & ncount>cpurei(5)
gt100	Go To statement label 100. set true if: succes == 0 succes == 1 & [iroute == 1 .and. .not.btest(imdctl(1),8)) .or. .not.btest(tsc(it)%tsppac,0)]
gt101	Go To statement label 101. small error time step doubler (if step was halved previously). set false initially & when .not.gt100 & dt<=dtmin_i & .not.gt19 Set true if succes.le.1 & [iroute.eq.1 .and. .not.btest(imdctl(1),8)) or .not.btest(tsc(it)%tsppac,0)] and .not.chngno(60)
gtrtn	Go To ReTurN statement. true if done/=0
i	Index to current time step card.
icard	fa-index of I-th (current) timeCARD.
icoran	bin Index to use for COuRANt limit (normally 2).
iecf	Integer Edit Control Flag. packed word for minor edits, major edits, & restarts edit control flag, 4 bits, numbered from right. Also used as a counter in the courant limit calculations.

Bit 1	(=1) call rstrec for restart edits
Bit 2	(=2) call majout for major edits
Bit 3	(=4) call mirec for minor edits
Bit 4	(=8) call pltrec for plot edits.
iosini	Indicator for On-line Semi-Implicit/Nearly-Implicit time advancement method.
iplt	temporary storage for iecf.
iprnt	PRINT Indicator, local value.
it	Index of Time-step card
ivskp2	some bits out of print.
j	pointer to next timestep card.
last	true when LAST advancement for this time step card.
nmon	Number of lines to write on MONitor before another label.
numdt	NUMber of DTmax steps between timehy and the current edit times (curtmi, curtmj, curtrs).
nwqa	pointer to second half of plot file scratch area.
rf	Reduction Factor for time step (normally 0.5).
ssdtim	Steady State TIMe step.
stsold	time for next write to screen.
SUCCES	Integer flag indicating SUCCESs of the advancement.
0	no need to repeat advancement with reduced time step
1	excessive truncation error
2	water property error
3	non-diagonal matrix
4	metal appears
5	air appearance, velocity flip-flop, or water packing advancement with same time step size
6	minimum volume of a variable volume
timeend	END of requested TIME step.
timlft	TIMe LeFT on this requested time step card, = timeend - timehy.

Note that this Table can be very useful when working with older versions of DTSTEP.

6.4.3 Mnemonically rename variables

Some of these variables, particularly the logical ones that started with “gt,” were still unclear as defined above in terms of logical operations. These were given mnemonic names in Table 5. It is then clear from the name what the variable means.

Table 5. Mnemonic names for logical variables

Name	Mnemonic Name	Meaning
gt14	lLoadOldTimeValues	Use subroutine MOVER to load old time values into current time arrays
gt19	luserInterface	The user interface is activated
gt27	lEditThenQuit	Perform all output (plot, minor/major edits, restart) then quit
gt38	lRepeatWSameDt	Repeat the advancement using the same timestep (happens when there is water packing, velocity flip-flop, etc.)
gt39	lDoEdits	Perform all ***selected*** output (among plot, minor/major edits, restart). Note: Variable iecf hold the selection info.
gt100	lDoMsErrChk	Do mass error checking
gt101	lDoubleDt	Double the timestep

The use of camelback notation, where the first letter of each word in the variable name is capitalized, helps clarify what the name means at a glance.

Some of the variables initially eliminated were aflag, htimehy, nwqda, nwqdaa, and zeitmi. Many logical variables were eventually eliminated also.

6.4.4 Fix the Declaration Section

The declaration section was reorganized in three ways. First, all the conditional code marked with pre-compiler directives were put after unconditional code. Second, all integer statements were pulled together into a groups as were all real, logical, and character statements. Third, the variables on each declaration statement were alphabetized.

The unconditional declaration section was broken into two parts, the non-integer variables, and the integer variables. Each of these parts had a data dictionary from Tables 4, 5, and 6. The only exceptions to this subdivision of declaration were references to new modules backmod and testmod, which were introduced for the integer algorithm and which must occur in the module section according to ANSI rules of Fortran 95.

6.4.5 New Integer-Algorithm Variables

Table 6 lists the local variables created to implement the integer timestepping.

Table 6. New integer timestepping variables created.

Key:	**Captialized letters in definition correspond to letters of name**
Variable Name	**Definition**
curtmin	CURrenT MINimum of 3 time targets: minor & major edits and restart
diff	DIFFerence between two quantities.
dtAdjTSave	DT ADJusted Time SAVE. Time target for next advancement unless overridden by Courant, stretch, or unusual time handling.
dtmax_old	value of DTMAX from the OLD timecard, initialized to 0.
dtmin_old	value of DTMIN from the OLD timecard, initialized to 0.
dtratio	DTmax/dtmin RATIO
dtsmallest	Δt_{small} from eq. 6b; mnemonic: SMALLEST usable DT.
endInterval	END points of timecard INTERVALs starting with the endpoint (which is normally 0).
endold	ENDtime from OLD (previous) timecard
endtimecard	ENDtime from current TIMECARD
Icardbgn	Integer BEGinning time of current 30imecard.
Icard_old	Index (fa) of previous time CARD.
Idt	Integer timestep corresponding to real DT from eq. 8a.
iduration	Integer DURATION of the current timecard. May differ from floating point duration because of unusual times.
Iedit	Integer time target for EDITting (plot, minor, major, rest and possibly message-exchange and end times)
iendtran	Integer time target, END of TRANSIENT.
Iendtime	Integer time target, END of CURent time interval (card).

Inttwo	INTeger TWO, 8-byte constant value 2.
Irall	Integer to Real difference relative test against dtsmALLest
iredit	Integer Relative EDIT times. (minor, plot, major, restart, message-exchange)
irtime	Integer Relative TIME. IRTIME .eq. ITIME – ICARDBGN.
Itemp	Integer TEMPorary variable for calculating intermediate quantities
itime	Integer equivalent of cumulative TIME, timehy
Itimecard	Index, within endInterval, of the user's timecard that is in control
kk	integer temporary variable for KalKulating intermediate quantities
kNonInt	unusual handler linked-step number. mnemonic is: Kalculate NON-INTeger steps
lstretch	Logical STRETCH logic active.
Msgtim	MeSsaGe TIMe. Data transmitted via PVM messages.
Nextdtmax	NEXT integer multiple of iDTMAX above cumulative time.
Nhalvings	Number of HALVingS necessary to reduce the maximum time step to as close as possible to the minimum w/o going below it.
Ntimecard	Number of TIMECARDs
prevAdvTimecard	PREVious ADVancement's TIMECARD ordinal.
supremum	the new timestep, which is a halving of dtmax, cannot exceed SUPREMUM.
tyme	time (cumulative) calculated from integers.

6.5 New Integer Timestepping Modules

Two new modules were created for integer timestepping for RELAP5-3D DTSTEP. The modules are summarized in Table 7.

Table 7. New integer timestepping modules

Name	Description
Backmod	Controls time step selection and frequency of output and plotting edits during transient advancement.
Testmod	Test Matrix Module. Data for logic path input and logic path test processing. Its initial design is for DTSTEP.

The variables of the Table 7 modules are summarized in Table 7a.

Table 7a. New integer timestepping modules

Name	Variable	Description
Backmod	lBackupOneDt	Logical indicator for BACKing UP ONE DT (time-step). Important for New Algorithm 6 – the unusual handler. .true. means back-up was performed in DTSTEP and was not advanced. .false. means back-up was not performed. It is set in DTSTEP(LoadOldTimeValues-subr). Used in TRAN.
Testmod	iTestDtstep	Controls the Test Matrix Function in RELAP5-3D DTSTEP 1 = force Courant dt halving 2 = force halving of timestep 3 = simulate PVM error condition 4 = force minimal dt
	ilowlimit	Activate the Test Matrix condition first on this timestep in subr. testDtstep.

	iuplimit	Deactivate the Test Matrix condition on this timestep in subr. testDtstep.
	testNo	In subr. rdbug, the Basic Test Case Number for the DTSTEP Test Matrix.
	Aset	Test Matrix character strings identifying A-set as either A1 or A2.
	ctest	Complete description of the Basic Test Case to run: (Test#)AABBCCDD Test# ranges from 1-17 AA ranges from A1 to A2 BB ranges from B1 to B3 CC ranges from C1 to C4 DD ranges from D1 to D7
	cbasic	Basic Test character identifier. Its values are "a1", "b1", "b2", "c1", "c2", "d1", "e1", "e2", "e3"

6.6 Internal Subroutines

Twenty internal subroutines were created. Some were abstracted from the code to clarify the logic flow of the main algorithm of DTSTEP. Others were created to have repeatable, reentrant code for use with the integer timstepping algorithm or for floating point or PVM-message operations. The subroutines are described in the following table.

Table 7b. New Internal Subroutines

Name	Origin	Description
astraTscCheck	Abstracted	This normally inert coding belongs to the IAEA Graphical User Interface (GUI) known as astra.
doubleDt	Abstracted	Only after an even number of advancements with the current timestep, double the floating point timestep and halve variable nrepet, which controls halving/doubling. (It has conditional experimental non-halving/doubling coding for multiplying by 1.1, but this is almost never used.)
dthyCalc	Abstracted	Reach the next floating point edit time target if it is within range if it is no more than 1.1*dtmax_i away from now, use one step: • Timecard (or transient) endtime is the ultimate stretch target • Never bypass the explicit exchange time. Also calculates some legacy DTSTEP variables that are currently not in use.
dtoutput	Abstracted	Perform all code output. Do edits according to the value of variable iecf. Also do output for interfaces and to the screen.
dtstat	Abstracted	This coding collects the statistical measures of Figure 3a, section 3 presented by major edits.
dtterm	Abstracted	This coding collects the termination conditions of Figure 3a, section 2.
dt60select1	Abstracted	This belongs to the experimental non-halving/doubling coding that is almost never used and is not part of the integer timestepping.
editTime60	Abstracted	This belongs to the experimental non-halving/doubling coding that is almost never used and is not part of the integer timestepping.
exp_cpl_int	Abstracted	Sends and receives the various types of explicit coupling messages. There is logic to send the following four types of explicit messages: parallel synchronous, parallel asynchronous, sequential synchronous and sequential asynchronous.
findHalving	Abstracted	This coding implements New Algorithm 5.

firstEntryOnly	***Created***	Implements New Algortithm 1. It creates array endInterval and initializes it to timecard endtimes. It initializes many new scalar variables for the integer timestepping.
get800x	Abstracted	For PVM Explicit messages 8002 through 8005, listen for and receive PVM-Executive's 8000 + x message (x = 2, 3, 4, or 5). *This routine was abstracted from 4 sections of nearly identical code in DTSTEP and reduced to a single subroutine.* Check if "my task ID" is the SEND-task-ID in the message. If so, send my data (if any) to the appropriate task(s) and wait for a reply. After a reply, possibly send again. If not, check if this task is to receive data from the SEND-task-ID. If so, receive it, store it, and send acknowledgement. Continue sending and receiving until all required data is sent. Then notify PVM-Executive, data transfer is finished. PVM-Executive send 8000 + x message with sndtid==0 to indicate that message type is finished (and the "Send or Receive Loop" may be exited).
initTargets	***Created***	Subroutine sets integer timestepping variables for the end of timecard and end of transient.
initIntTime	***Created***	Implements New Algorithm 4 and sets Δt_1 to $\Delta\tau_{max}$.
intCardTargets	***Created***	Implements New Algorithm 2, 3, and 4 either via direct calculation for synchronous coupling, or else through calls to updateCardTarget. Also, calculates integer equivalent to end of previous timecard and end of current timecard.
intTarg8000	Abstracted	This coding collects the Integer target operations conducted in the 8000-section of dtstep, combining elements of New Algorithms 2 and 4. Some integer time quantity updates are performed only if coupling is syncronous.
LoadOldTimeValues	Abstracted	For the case of a code advancement back-up, move values from “old time” arrays, that hold values from the previous advancement, into the current time arrays.
proc_8000	Abstracted	Processes the data associated with the 8000 message from PVMEXEC. It determines if the message should be exchanged using the check_8000 function from the shared module. If appropriate, the message is received, the data are unpacked and the acknowledgement message is sent back to PVMEXEC.
proc_8001	Abstracted	Sends the 8001 message with a requested timestep size for synchronously coupled calculations. The subroutine then receives the floating point timestep size and the values for curtmi, curtmj and curtpl from PVMEXEC. New values of iredit are calculated based on the received edit times.
updateCardTarget	***Created***	If edit time has been reached, update edit times using user-specified frequency. This implements New Algorithm 4, Eq. 7b for a single value of j.

The following internal subroutines were abstracted by moving single-entry code and adjusting it to work correctly as a subroutine:

dthyCalc, dtterm, dtstat, LoadOldTimeValues, doubleDt, dtoutput, editTime60, Abstracted dt60select1, proc_8000, proc_8001.

The following internal subroutines were constructed by abstracting coding used in more than one place to make multiply-called reentrant coding:

- findHalving – used in two places to find an acceptable Δt at or below a target.
- Get800x – abstracted from 4 places to reduce coding into a single efficient routine.

The following internal subroutines were created to implement new integer algorithms or to incorporate integer algorithm coding. Subroutines dthyCalc and findhalving are listed here because they implement New Algorithms for integer timestepping.

dthyCalc, firstEntryOnly, initTargets, initIntTime, intCardTargets, updateCardTarget, findHalving, intTarg8000.

7.0 Implementation in PVMEXEC DTSTEP

The authors of PVMEXEC DTSTEP created it from a copy of the RELAP5-3D DTSTEP subroutine and adjusted as needed to function as the timestep controlling executive.

7.1 Incorrect vestiges of RELAP5-3D in PVMEXEC DTSTEP

Before the implementation of integer timestepping, PVM DTSTEP, the subroutine was studied and its functions identified. In the process, it was found that there were many comments, variables and some legacy code that had nothing to do with the function of the executive and were not used. Most were vestiges of RELAP5-3D DTSTEP. These confused the conversion to integer timestepping, development and debugging. There was also the possibility that they could inadvertently cause errors.

The following describes sections of the PVMEXEC DTSTEP subroutine that were either unnecessary, too complex, or perhaps incorrect (at least, it did not match the manual). Some comment statements were misleading or incorrect. Items are presented in the order that the statements or sections occur in base PVMEXEC DTSTEP, version 2.4.1.2. All were appropriately corrected as indicated with *italicized* remarks.

(1) On termination check (pvm_succes==6), DTSTEP does the following:
. set done=-9, iecf=15, gt27=T, go to 27
. set gt27=gtrtn=F, iplt=iecf, skip msgs 8002-8005 processing, gtrtn=T, go to 900
. skip: if gt27==T, go to 27, finally RETURN becase gtrtn==T.
Since nothing is done with iecf, just return after done=-9.

(2) Section "Come here for a good advancement" is overly complex.
Eliminate gt100 and label 801; just "go to 100" instead of 801.

(3) If timestep < minimum, do not stop; rather double timestep & continue
Actual statement: if (dt .le. dt_min(i)) go to 70
This is RELAP5-3D DTSTEP logic, but should be terminal in PVMEXEC.

(4) "Load old-time values for halved time-step" comment is meaningless.
Comment was removed.

(5) Failure conditions (pvm_success>=2) are checked AFTER timestep doubling.
This should be checked right after termination test (pvm_succes==6)

(6) k = pvm_succes - 2 unneeded (from RELAP5-3D)
Just compare pvm_succes to 2.
(7) The following comment is meaningless because there is no variable errmax.
"C If errmax is small, double halved time-step"
Comment was removed.
(8) Comment "Do requested edits" above statement 27 is misplaced.
What follows 27 is messages 8002-8005 processing. Editing occurs after that the comment "send global print control to processes"
(9) Section "Finished with this transient?" should be in the advancement above the 8002-8005 section because messages 8002-8005 do not change variable done.
(10) ** At the end of time interval, the last two steps are supposedly equal.
This condition is not enforced UNLESS nrepet==0 (dt==dtmax(i)).
This is not what the manual says.
(11) The following comment is meaningless because there is no Courant limit in the Executive.
"c Set dtx to the minimum of courant, control, and 110% old value."
Removed comment.
(12) The following comment is meaningless because there is no 105 card in PVMEXEC:
"c Check if advancement is to be terminated by 105 card input."
Removed comment.

7.2 Definitions, elimination of vestigal variables

During the initial study of PVMEXEC DTSTEP, the variables were studied and defined to help understand the logic flow. Unnecessary variables and code were marked and eventually eliminated during the conversion process. Such variables are marked by asterisks in Table 8.

Table 8. PVMEXEC DTSTEP variables with potentially unneeded variables marked

Name	Definition
af	Amplification Factor for time step (normally = 2.0).
contrl_timestep	CONTRoL TIMESTEP
curctl	CURrent ConTroLling time step card number
curtex	CURrent time target for EXplicit message
curtmi	CURrent time target for MInor edit
curtmj	CURrent time target for MaJor edit
curtpl	CURrent time target for PLots
curtrs	CURrent time target for RsStart
done	0 means not DONE, negative values indicate error conditions
dt	Delta Time, the current time step.
dtadj	the ADJusted DT (doubled/halved).
dtctr	ConTRol DT. it is a parameter = 10**75. Could be replaced.
dt_control	DT CONTROL information, an array with length n_timesteps, its bits control the time stepping for each timecard.
dthy	HYdraulic DT. minimum of new adj dt and last time step value of dtx.
**dthyx	HYdraulic DT eXtra copy. not used. Should be deleted.
**dtht	HeaT DT. set but never used. Should be deleted.
**dtintv	DT INTeRval for minor/plot, major, or restart edits.

**dtrem	Delta Time (cpu_limit – timehy) REMaining on cpu clock for this problem.
dtx	courant DT.
**dtxmdt	?? declared in DTSTEP but not used. Should be removed.
edittime	minimum of approaching EDIT TIME targets.
endtime	END TIME of current timecard. right endpt of time interval
end_time	array of END TIMEs of all the timecards.
**exflag	EXplicit FLAG.
exp_cpl_freq	FREQuency of EXPlicit CouPLing message exchanges.
fail	FAILure indicater. true means code failure; quit.
**flag	true if we still need to do some steps to get to dtmax. ** always has value "nrepet.eq.0". Should be replaced.
gt100	Go To 100. true if-and-only-if pvm_succes .eq. 0 At label 100 "if errmax small, double halved time-step"
gt101	Go To 101. when true, decrement nrepet. Label 101 is within the Label 100 section.
**gt14	Go To 14. ALWAYS FALSE and should be removed.
**gt19	Go To 19. true only if pvm_succes .eq. 2.
gt27	Go To 27. true only if pvm_succes .eq.6. Label 27 starts the "Do requested Edits" section.
**gt38	Go To 38. ALWAYS FALSE and should be removed.
**gt39	Go To 39. ALWAYS FALSE and should be removed.
**gt39a	Go To 39. ALWAYS FALSE and should be removed.
gtrtn	Go To ReTurN
**help	HELP variable from RELAP5-3D. declared in DTSTEP but used nowhere in PVMEXEC.
**htimehy	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
**itimehy	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
**i	index to current time step card. **Same as curctl**.
**icard	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
**icoran	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
iecf	packed word for minor edits, plots, major edits, restarts edit control flag, 4 bits, numbered from right 1 (=1) call rstrec for restart edits 2 (=2) call majout for major edits 3 (=4) call mirec for minor edits 4 (=8) call pltrec for plot edits.
info	integer*4, INFOrmation necessary for PVM.
iplt	temporary storage for iecf?
**iprnt	never used and should be eliminated. local value of print.
**iv	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
**ivskp2	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
**iwrd8	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
j	pointer to next timestep card.
last	true if this is the last adv. for this time step card.
lexplt	Logical, EXPLiciT message exchange control
lminor	Logical, MINOR edit message exchange control

lmajor	Logical, MAJOR edit message exchange control
lplot	Logical, PLOT message exchange control
lrst	Logical, ReSTart message exchange control
**messg	from RELAP5-3D, declared in DTSTEP but used nowhere in PVMexec.
mip_freq	MInor edit/Plot FREQuency (number of dt_max timesteps major edits.)
mj_freq	MaJor edit FREQuency (number of dt_max timesteps between major edits.)
msgtag	MeSsaGe TAG for pvm.
**nbyte	declared in DTSTEP but used nowhere in PVMexec.
ncount	Number or COUNT of time step advancements
nitem	integer*4, Number of ITEMs in a message.
**nmechk	No Mass Error CHecK, from RELAP5-3D. set but never used and should be removed.
**nmon	declared in DTSTEP but not used.
nrepet	Number of REPEaTs of current time step to reach next multiple of DTMAX.
**nstsp	Set but never used. Should be removed. set to zero and incremented or decremented but not output or in any test or calculation.)
**nwqa	declared in DTSTEP but not used anywhere in PVMexec.
**nwqda	declared in DTSTEP but not used anywhere in PVMexec.
**nwqdaa	declared in DTSTEP but not used anywhere in PVMexec.
n_exsync	Number of EXplicit SYNChronously coupled processes
n_exasync	Number of EXplicit ASYNChronously coupled processes
n_excsync	Number of EXplicit SYNChronous sequential coupled processes
n_excsync	Number of EXplicit ASYNChronous sequential coupled processes
n_proc	Number of coupled PROCesses
n_synch	Number of SYNChronously coupled processes
n_timesteps	Number of TIMESTEP cardS
**numdt	integer NUMber of DTmax steps between timehy and the current edit times (curtmi, curtmj, curtrs). never used. Should be removed.
**prevnd	PREVious time step card eND time. Set but never used. Should be removed.
procdt	The proposed DT by the synchronously coupled PROCesses
proc_rpt	list of PROCesses RePorT statuses 0 not reported 1 reported
proc_sync	list of PROCesses coupled SYNChronously
pvmdt	PVMexec selected DT = minimum value of all time steps proposed by synchronously coupled processes.
**pvmint4	INTEGER4
**pvmreal8	REAL8
pvm_succes	SUCCESs of the PVM coupled calculation at current step. 0 means continue 1 means running at minimum time step is okay x for 1 < x < 6 are various time step controls 6 means stop the coupled calculation
rf	Reduction Factor for time step (normally 0.5).
rpt_stat	RePorT STATus of all processes
	0 = not all process reported

	1 = all processes reported
rst_freq	ReSTart FREQuency (number of dt_max timesteps between restarts.)
skipt	SKIP certain coding when True (.true. when the calculation is a restart)
sndtid	Task IDentification to SeND
stride	integer*4, STRIDE (skip factor) through the data.
**stsold	declared in DTSTEP but used nowhere in PVMexec.
tid	integer*4, Task IDentifier.
timehy	cumulative TIME HYdraulic.
**time4	declared but never used.
**timlft	time left on this time step card, tpsend(i)-timehy. Set but never used. Should be removed.

Of the variables marked with ** in Table 8 as being potentially unneeded at the early stage of the project, 34 were eliminated. They are:

dtht, dthyx, dtintv, dtrem, dtxmdt, exflag, flag, gt14, gt19, gt38, gt39, gt39a, help, htimehy, icard, icoran, iprnt, itimehy, iv, ivskp2, iwrd8, messg, nbyte, nmechk, nmon, nstsp, numdt, nwqa, nwqda, nwqdaa, prevnd, stsold, time4, timlft

The three, that were kept because they were found to be necessary, are:

i, pvmint4, pvmreal8

7.3 Reworking source code of PVMEXEC DTSTEP

The source code was made more readable by eliminating unused variables and code, inserting a data dictionary, inserting numerous, outline-style comments, and regrouping sections of code. This aided in the conversion and debugging processes.

7.3.1 Deleted source code from PVMEXEC DTSTEP

Coding that was removed includes:

- All declarations of unused variables that were eliminated in Sec. 7.2
- All statements that used unused variables that were eliminated in Sec. 7.1
- Some code that was unreachable because some logical variables were always false

7.3.2 Declarations in PVMEXEC DTSTEP

The declaration statements were grouped as follows:

- modules in alphabetical order (except for machine)
- other global variables (include statements and common block dtstepc)
- local variables.

Among global and local variables, the declarations are grouped by type; the groups are ordered:

- Real declarations
- Integer declarations
- Logical

Variables were alphabetized on declaration statement to aid in finding them, except for the common statements which induces an order.

The local declaration section was further subdivided into original and integer timestepping variables. The original section was divided into active and eliminated. The declaration of all variables that were eliminated in Sec. 7.1 were collected and marked as junk. These were later removed altogether after verifying that they had no impact on code performance.

All unused sections of declarations marked with pre-compiler directives were eliminated. These three sections are:

bufr, cmprs, npa

All declarations of unused variables, those that were marked in Table 7 with asterisks were also removed.

7.4 Internal documentation in PVMEXEC DTSTEP

An outline style of numbered comments, using the 1, 1.1, 1.1.1, system for subsections was introduced to break out major sections and progressively minor subsections. Explanations of operations were expanded from 0-2 lines to whatever amount was deemed necessary to explain a section to a developer who is completely unfamiliar with DTSTEP. This was helpful in the development and debugging stages.

During the development and debugging, sometimes the comments became incorrect. Attempts were made to keep them correct; however, this was not fully successful.

7.4.1 Data dictionary

There are two data dictionaries:

- original variables
- integer timestepping variables.

The variables are listed in alphabetical order with its definition using a capitalization key to indicate the mnemonics of the name.

The original variable data dictionary was constructed from Table 8. Those variables that are not marked with asterisks were eventually the only ones remaining in the data dictionary. The integer timestepping variables data dictionary follows the declaration for the original variables in DTSTEP.

7.5 Integer Timestepping in PVMEXEC DTSTEP

The implementation involved creating and documenting the new variables for timestepping, coding the integer timestepping New Algorithms, and breaking out internal subroutines. The latter is covered in the next section.

7.5.1 Local integer timestepping variables

The new variables introduced to PVMEXEC DTSTEP to implement the integer timestepping are given in Table 9.

Table 9. Integer Timestepping Local Variables in PVMEXEC DTSTEP

dtsav	DT SAVe. Copy of dt made before special timecard endtimes
dt_targ	DT TARGet. dt made to handle special timecard endtimes to override pvmdt as

	requested by coupled codes.
icardkind	Integer timeCARD-defined frequency of given KIND of edit (plot, restart, etc)
idtsav	Integer DT SAVe. Copy of idt made before special timecard endtimes
idt_targ	Integer DT TARGet. idt made to handle special timecard endtimes used to override pvmdt as requested by coupled codes.
iredit	Integer Relative EDIT time, $\tau_{E,k}$ of Eq. (8b).
itime	Integer cumulative TIME, T/ Δt_{small}.
itmp	Integer TeMPorary
itsav	Integer Time (cumulative) SAVe. Copy of timehy made before handling special timecard endtimes
ladd(5)	Logical Array of Dump time inDicators. (Not used) (1) lminor, (2) lplot, (3) lmajor, (4) lrst, (5) lexplt
lBackupOneDt	Logical indicator to BACKUP ONE DT (time-step). .true. means do a backup
lrestore	Logical RESTORE flag. when TRUE it means that dt and idt are to be restored to dtsav and idtsav and ovderride the pvmdt requested by the coupled codes.
supremum	the new timestep, which is a halving of dtmax, cannot exceed SUPREMUM.
tdiff	Time DIFFerence. Difference between floating point time and time calculated from integers.
tmpmax	integer TeMPorary MAXimum used to calculate idtmax.
tsav	Time (cumulative) SAVe. Copy of timehy made before handling special timecard endtimes.
tyme	Ti(Y)ME calculated from integers for diagnostics.

7.5.2 Module integer timestepping variables

A new module for integer timestepping variables was created called TARGETMOD. It contains data for the PVM Executive's integer time targets that specify the following:

(1) End of transient
(2) End of time for the current timecard
(3) Explicit message exchange time
(4) Major edit time
(5) Minor edit time
(6) Plot time
(7) Restart dump time

The module variables are given in Table 10.

Table 10. Integer Timestepping Local Variables in TARGETMOD

dtsmallest	SMALLEST allowable value of DT, Δt_{small} from New Algorithm 1, Eq. 6b
endInterval	END time of a time INTERVAL (right endpoint). endInterval(0) = "start time of case"
hitend	HIT the END of timecard. true means close enough to hit end by adjusting dt (compressing or stretching up to 110%)
icardbgn	Integer BeGiNning time of current timeCARD in terms of dtsmallest. $T_{C-1}/ \Delta t_{small}$.
icardend	Integer timecard END time of current CARD in terms of dtsmallest. $(T_C - T_{C-1})/ \Delta t_{small}$.
icardtime	Integer TIME accumulated since start of current timecard. $\tau_{C,i}$ of Eq. (7).
idt	Integer DT, $\Delta\tau_i$ in New Algorithm 4, Eq. (8a)

	# of Herz or relap5-cycles that equal dt.
idtadj	Integer DT ADJustment. Not used (yet)
iduration	Integer DURATION of the current timecard = icardend - icardbgn. May differ from floating point duration because of unusual times.
iloc	i-value LOCating the minimum dt_min among timecards
idtmax	Integer largest value of DTMAX. $\Delta\tau_{max} = 2^H$ in New Algorithm 1, Eq. (6a)
iedit	Integer EDIT time
iendtime	Integer END TIME for the entire transient
iexplfrq	Integer EXPLicit FReQuency. exp_cpl_freq
iexpltime	Integer EXPLicit TIME. relap5-cycle for exchanging explicit coupling data
iloc	I-value LOCating the minimum dt_min among timecards.
imajorfrq	Integer MAJOR FReQuency. mj_freq
imajortime	Integer MAJOR TIME. relap5-cycle for next major edit
iminorfrq	Integer MINOR FReQuency. mip_freq
iminortime	Integer MINOR TIME. relap5-cycle for next minor edit
iplotfrq	Integer PLOT FReQuency. mip_freq
iplottime	Integer PLOT TIME. relap5-cycle for next plot
irestfrq	Integer RESTart FReQuency. rst_freq
iredit	Integer Relative EDIT times. These are restarted at the beginning of each timecard. 1 = minor edit, 2= plot, 3= major edit, 4 = restart, 5 = explicit coupling message time
kk	integer temporary variable for KalKulating intermediate quantities
nhalvings	Number of HALVINGS that reduces DTMAX to dtsmallest. H in New Algorithm 1, Eq. 6
prevAdvTimecard	index among user TIMECARDs of the PREVious ADVancement
spstat	SPecial STATus. Indicates status of processing a special endtime. 0 not at an unusual time target, proceed normally. 1 complete final step of an unusual time target; the first step (spstat.eq.2) was taken on the previous step. Hereafter, time is at a normal halving/doubling point. 2 timestep exactly reaches unusual time target. 3 (may not be taken) timestep is half the distance to the unusual time target.
iresttime	Integer RESTart TIME. relap5-cycle for next restart file write
itime	Integer TIME

7.6 Subroutines introduced for restructuring

As with the creation of the PVMEXEC DTSTEP from a copy of the RELAP5-3D DTSTEP subroutine, the internal subroutines were initially copies of those in RELAP5-3D DTSTEP. They were then adjusted as needed. This exactly follows the original methodology employed by the authors of the PVM Coupling coding.

A significant effort was made to restructure the code during the resolution of this issue. Primarily, this has been accomplished by taking the previous *dtstep* subroutine and reducing it to the high-level logic path with a significant number of “contained” subroutines. This restructuring is most noticeable in the PVMEXEC *dtstep* subroutine. While the restructuring of the code did not solve any problems, it did provide additional clarity as to the overarching code logic, which did help in providing a better understanding of the algorithm which ultimately led

to a successful implementation of the integer-based timestep size selection and advancement algorithm.

Fourteen internal subroutines were created. Some were abstracted from the code to clarify the logic flow of the main algorithm of DTSTEP. Others were created to have repeatable, reentrant code for use with the integer timstepping algorithm or for floating point or PVM-message operations. The subroutines are described in the following table.

Table 7a. New Internal Subroutines

Name	Origin	Description
firstEntryOnly	**Created**	On the very first entry to DTSTEP (and not thereafter), perform initializations and allocations. Find minimum time-step on all timecards. Differences with RELAP5-3D counterpart, firstEntryOnly: No endInterval array and fewer initializations.
explicit_async_ parallel_exchange	**Abstracted**	This subroutine determines if this is the appropriate time to perform an explicit, parallel, asynchronous data exchange between any of the coupled codes. If it is, each of the codes that should be sending data is notified. The subroutine then waits for acknowledgement that all of the receivers have obtained all of their data. Finally, this subroutine sends the "all clear" message to all of the explicit, asynchronous, parallel coupled jobs to permit them to resume their calculations.
explicit_async_ seq_exchange	**Abstracted**	This subroutine determines if this is the appropriate time to perform an explicit, sequential, asynchronous data exchange between any of the coupled codes. If it is, each of the codes that should be sending data is notified. The subroutine then waits for acknowledgement that all of the receivers have obtained all of their data. Finally, this subroutine sends the "all clear" message to all of the explicit, asynchronous, parallel coupled jobs to permit them to resume their calculations.
explicit_sync_ parallel_exchange	**Abstracted**	If a simulation includes explicit, synchronous, parallel coupling, each of the codes that should be sending data is notified. The subroutine then waits for acknowledgement that all of the receivers have obtained all of their data. Finally, this subroutine sends the "all clear" message to all of the explicit, asynchronous, parallel coupled jobs to permit them to resume their calculations.
explicit_sync_ seq_exchange	**Abstracted**	If a simulation includes explicit, synchronous, sequential coupling, each of the codes that should be sending data is notified. The subroutine then waits for acknowledgement that all of the receivers have obtained all of their data. Finally, this subroutine sends the "all clear" message to all of the explicit, asynchronous, parallel coupled jobs to permit them to resume their calculations.
findHalving	**Abstracted**	This coding implements New Algorithm 9. Moved into the shared module, idtmod; see Section 8.0.
intCardTargets	**Created**	Subroutine implements New Algorithm 4, by calling subroutine updateCardTarget five times, once for each type of edit time target. Differences with RELAP5-3D counterpart, intCardTargets: This subroutine is extremely simple by comparison, because there are no special cases for PVM vs. non-PVM operations, nor for synchronous vs. asynchronous. Differences with RELAP5-3D counterpart, outputInts:

		This one uses array iredit rather than mnemonically named scalars.
intDtMax	**Created**	This ***function*** subprogram implements New Algorithm 1.
outputInts	Abstracted Deprecated	Subroutine implements New Algorithm 4, by calling subroutine updateIntTarg five times, once for each type of edit time target, using mnemonically named scalars. Differences with RELAP5-3D counterpart, intCardTargets: This subroutine is extremely simple by comparison, because there are no special cases for Pvm vs. non-PVM operations, nor for synchronous vs. asynchronous. This subroutine was replaced by intCardTargets below.
process_8000	**Abstracted**	This subroutine processes the data associated with 8000 message. It determines if the message should be exchanged using the check_8000 function from the shared module. If appropriate, the message data are packed and sent to the appropriate coupled code. The acknowledgement message for message tag 8000 is also received.
rec_8001	**Abstracted**	This subroutine receives the 8001 message, which contains the requested timestep size, from each of the codes that are synchronously coupled.
timeCardInit	**Created**	Subroutine called at the end of timecards to implement New Algorithm 2, 3, and 4 in setting up next timecard's control. It finds dtsmallest from the next dtmax and dtmin, resets integer timecard time to zero, and reset all integer time targets. Differences with RELAP5-3D counterparts, initIntTime & intCardTargets: It performs many more initializations than initIntTime. Unlike IntCardTargets does not call updateCardTarget to initialize time targets; it also uses mnemonic scalars also for integer time targets.
updateCardTarget	**Created**	If edit time has been reached, update edit times using user-specified frequency (RELAP5-3D Manual Volume 2, Appendix A). This implements New Algorithm 4, Eq. (8b) for a single value of j. Also implements the 110% stretch logic of Sec. 5.4. Differences with RELAP5-3D counterpart, updateCardTarget: The user-specified frequency can be zero; so there is special coding to handle that. Also, the stretch logic is not present in the counterpart.
updateIntTarg	**Created** Deprecated	Subroutine implements New Algorithm 4, Eq. (8b) for one value of j. It converts a timecard edit frequency (RELAP5-3D Manual Volume 2, Appendix A) to an integer frequency, ifreq, carefully. Differences with RELAP5-3D counterpart, updateCardTarget: It performs the rounding conversion that RELAP5-3D DTSTEP internal subroutine intCardTargets does in addition to the minimization adjustment of updateCardTarget. This subroutine was replaced by updateCardTarget below.

Among these subroutines, most were created specifically to implement the integer algorithm.

8.0 Shared Module - idtmod

The PVMEXEC and RELAP5-3D computer programs both have similar tasks associated with the selection of timestep sizes to hit certain specified communication points. However, many

different routines or techniques were used to perform the same task. The use of different implementations of what should have been the same logic created scenarios where the inconsistencies, while maybe minor (including different order-of-arithmetic issues), were sufficient to preclude proper execution of the problems. To remedy this, a shared module, *idtmod*, was created. This module contains both data and sub-programs that are needed by both PVMEXEC and RELAP5-3D.

A number of quantities and functions are exactly the same between the two subroutines R-DTSTEP and P-DTSTEP. Many variables and utility subroutines were identified, renamed to be consistent between the DTSTEP subroutines, and collected into the shared module.

8.1 Data

The shared module has been developed to contain a small amount of data that is needed by the integer-based algorithm. The variables that are included in the module are detailed in Table 11.

Table 11. Integer Timestepping Local Variables in TARGETMOD

Logicals	
exp_cpl	Specify if EXPlicit CouPLing is active
ldbg	Logical DeBuG flag. When true, it activates additional debug print statements
Integers	
i8kind	Integer 8-byte KIND. Specifies 64 bit integer values as need by the algorithm
qdkind	QuaD-precision KIND. Specifies 128 bit floating point values
dpkind	Double-Precision KIND. Specifies 64 bit floating point values
dbgf	DeBuG File. The unit number used for the debug print statement
idt	Integer Delta-T. Representation of the timestep size. For normal timesteps, dt=idt*dtsmallest
idtmax	Integer DT MAXimum. Integer representation of the maximum timestep size
itime	Integer cumulative TIME. Integer counter for elapsed time
iendtime	Integer END TIME of the timecard.
iendtran	Integer END of TRANsient. Integer representation of the transient end time
icardbgn	Integer timeCARD BeGiN time. Integer representation for the start time of the current timecard
icardend	Integer timeCARD END time. Integer representation for the time equal to the end of the current time card
irtime	Integer Relative TIME. Integer representation of time elapsed from start of current timecard
iduration	Integer DURATION. number of timestep of size dtsmallest in the current time card
iredit(5)	Integer Representation of EDIT times
iedit	Integer representing the nearest EDIT point
Extended Precision Reals (128-bit floating point)	
qdtsmallest	QuaD-precision SMALLEST attainable DT. Floating point timestep size that is greater than DTMIN and is determined via Eq (1)
Reals (64-bit floating point)	

dtsmallest	SMALLEST attainable DT. Smallest floating point timestep size that is greater than DTMIN and is determined via Eq (1)
af	Amplification Factor. Factor that is used to increase the timestep size.
rf	Reduction Factor. Factor that is used to reduce the timestep size
dtadj	DT ADJustment.
curtmi	CURrenT next MInor edit time
curtmj	CURrenT next MAjor edit time
curtpl	CURrenT next PLot time
curtrs	CURrenT next ReStart time
endtime	problem END TIME
edittime	the next EDIT TIME target (of any kind, major, minor, etc.)
dtlimit	DT upper LIMIT based on error mitigation
dtsmallest	SMALLEST attainable DT. The smallest timestep size that is greater than DTMIN and is determined via Eq (1)

8.2 Module Internal Subprograms

These subroutines were abstracted from R-DTSTEP and/or P-DTSTEP. They were modified to work appropriately for both, such as adding a starting point to subroutine findhalving. In some cases, further improvements were made in resolving user problems, such as adding the creation of qdtsmallest to mkdtsmall. These subroutines were then unneeded in R-DTSTEP and P-DTSTEP and were removed.

Table 12. Integer Timestepping Local Variables in TARGETMOD

calctimehy	real function	Implements New Algorithm 3 to calculate the time given an integer value of time relative to the beginning of the timestep card.
check8000	Logical function	Determines if the 8000 message needs to be exchanged.
findhalving	Subroutine	Implements New Algorithm 9 to find the largest timestep size smaller than a given the value (provided in the calling sequence) by successively halving a starting value. The start value may be ΔT_{MAX} or an unusual timestep size.
isnormaldt	Logical function	Applies New Algorithms 7 and 8 to determine if the current timestep size is normal. A timestep is declared to be normal if the floating point value of the timestep size and the value that is calculated using Eq (1) differ by no more than 10^{-10} seconds.
makeconsistent	Subroutine	Applies New Algorithm 5 to reset the floating point representation of both the timestep size (dt) and the time (timehy) based on their integer equivalents for normal timestep sizes.
mkdtsmall	Subroutine	Implements New Algorithms 1 and 2 to calculate the variables dtsmallest and qdtsmallest based on the DTMIN and DTMAX.
movedtdat	Subroutine	This subroutine is used to advance or reduce both the floating point value of time (timehy) and the integer representation of time (irtime). If the timestep is normal, the code alters the integer representation of time and then uses the calctimehy function to calculate timehy. If the timestep size is unusual, the value of timehy is modified and the integer representation of the new time is calculated using the

		time2int function.
time2int	Integer function	Applies New Algorithm 4 to calculate the integer representation of a floating point value of time. The returned value is relative to the beginning of the timestep card.

Handling this module during maintenance and installation introduces a new nuance. It is a huge maintenance difficulty to maintain copies of the same file in two directories because eventually one copy or the other is changed without the other. The use of a Unix or Linux link can mitigate the issue; however, links can be broken. It is better to have just one file.

With the decision to retain just one copy of module idtmod, the installation process was affected. The module can reside in either the RELAP or the PVMEXEC directory, but not both. During installation, either linking or copying could provide the other directory with the files it needs. This was implemented.

9.0 Development and Debugging

The elements of integer timestepping described in Sections 2, 3, and 4 were developed early, while those in Section 5 were developed during the development. Similarly, all internal subroutines and most new variable names were created during the development process.

Development was performed in small increments because of the requirement to not change answers (except for correcting errors and for slightly improved time tracking). It was found that when numerous conversions were made, the likelihood of altering calculations increased. Therefore, changes were introduced in small groups, sometimes as small as three or four statements. After introducing changes, the complete set of standard installation problems, including PVM problems were rerun and compared against the base code output.

Eventually, the integer timestepping results began to differ slightly from the original due to more accurate time tracking. From this point onward, a second base case was established and output from developmental code was compared against it. This occurred and was repeated several times in the development process. This process is summarized as .

Incremental Development Strategy

1. Develop means to implement small, independent change
2. Code the few changes
3. Run all test cases
4. Compare results
5. If differences are found, decide if there is an error, or improved accuracy
    a. If error, determine source and return to 1.
    b. Otherwise, done

9.1 Development Issues

The two issues causing the greatest challenge to development were:

1. The problem of finding a suitable coding expression for an integer concept
2. Control coding

The issue of finding a suitable expression of an integer concept in coding is not new. It happens that coding something one way causes an error in one or more test cases, but that an alternative *programming-equivalent* set of coding runs correctly. This is sometimes revealed by placing the RELAP5 code on another platform and running the test set, but more often occurs on the developmental platform. The former is a well-known problem caused by differences in compilers, different release levels of the same compiler, math libraries and other operating system differences, and even in PVM interaction with the operating system level. The latter is caused by glitches in the compiler and in PVM interaction with the operating system.

The result is that implementation of the integer concepts was often not as straightforward as it would seem. For example, the ans79.i test problem uses huge timesteps. This caused New Algorithm 1 to fail when 32-bit integers were used, but once this was traced and understood, the problem worked properly when 64-bit integers were used. Other examples include the form of integer if-tests and the updating of time targets.

The second development issue is control coding. In many places, the means of updating time-related quantities was dependent upon whether RELAP or PVMEXEC was in control, or when PVM was in control, the kind of coupling was in use, particularly synchronous vs. asynchronous. In some places it was necessary to completely separate the control code on these bases, and in others it was not. For clarity, consistency and efficiency of coding though, redundancy is undesirable. Control coding required numerous reworkings for this reason.

Finally, Parallel Virtual Machine (PVM) exhibits different behavior on different platforms. These failures were caused by the inability of the operating system to obtain a PVM daemon when requested, which mostly occurred if PVM jobs were run in succession in a script. This made it appear that PVM test cases were failing when they were not and a rerun would actually work. Optional “permission to terminate” source coding was created to address this and PVM scripts had “post-run” delays and other correction coding added. These devices reduced but did not completely eliminate the problem.

9.2 Testing and Debugging

In order to assess the correctness of the integer time-stepping, implementation the existing set of test cases was augmented considerably of the transmittal set. An initial problem, time8, with 8 timecards that tested different aspects of timecard handling by RELAP5-3D run under PVMEXEC control was the first addition. Input files associated with User Trouble Reports (UTR) were also added; this included some problems that mimicked actual UTR when the original input files were unavailable. Some special cases, designed by Dr. Walter Weaver, that create and test failure handling in coupled problems were also incorporated. Finally, a huge test matrix that tests every kind of coupling and many other aspects of coupled calculations was constructed.

The so-called Weaver test set is presented in Section 8.2.1, the tests related to UTE in 8.2.2, and the so-called DTSTEP Test Matrix comprises Section 9.

9.3 Debugging: The Weaver test set

The so-called Weaver test set consists of 9 test cases designed to check various combinations of the DTSTEP time-step repeat logic and the ability of the code to detect errors and shut down without creating a hung machine condition.

Code modifications were necessary in subroutines VFINL and TRAN. The coding necessary for these tests is marked with pre-compiler directives testpvm1 through testpvm9. The entire RELAP5-3D code must be recompiled separately for each test case. This is automated in the controlling Linux script named RUNW.

No errors were found in DTSTEP as a result of these tests.

9.4 The TestDt Tests

The next group of tests had two components: available DTSTEP UTR input and the attempted recreations of unavailable UTR input. These input models are located in the run/TestDt directory of the RELAP5-3D source distribution. The Linux script created to run the test set is named runv_dt. The problems are listed in Table 11.

Note that files in Table 11 with the "ii" extension are the name of the PVMEXEC master input file; there are matched input files for the RELAP5-3D processes that the executive spawns. These input file names start the same but differ on the last letter; their extension is "i." For example, time8.ii spawns a process whose input file is time8p.i.

All available models in Table 11 ran with version R3D244B. Although input pvmcore6_20x.ii, unusual.i, and r5_melcorx.ii helped pinpoint errors in the implementation and eventually ran, other recreations failed to reproduce the reported behavior after repeated modifications and were thus unsuccessful in advancing the debugging and testing process.

The pvmcore6_20x.ii problem has 6 connection between master and slave in a semi-implicit coupling was of interest for both the impact of FORTRAN 95 conversion, a simultaneous project that was in its final stages, and for the ability of the BPLU solver to handle that many connections. It was used to help solve a RELAP5-3D/MELCOR coupling error wherein the restart at 10 seconds did not output the same values that the continuous run to 20 seconds had.

Table 13. TestDt input models for debugging unavailable input models

Name	Description
Section 1	**Available/Extended input models**
ed_UP04028.i	Input model for User Problem 04028
vhtr03009.i	Input model for User Problem 03009
time8a.i i	A fast-running version of time8.i
time8.ii	The original 8-timecard input model described in Sec. 8.2
unusual.i	A fast-running problem with a 4 different challenging unusual time issues
Section 2	**User Trouble Report mock-ups**
r5_melcorex.ii	Attempt to recreate the RELAP5-MELCOR problems reported in July 2009 Slave process input: r5_melcoreb1.i, and r5_melcoreb2.i
r5_cobra1109x.i	Attempt to recreate a RELAP5-3D coupled to COBRA calculation that hangs at 1107

	sec
Restart2ndx.ii	Attempt to recreate the problem with a restart of a restart of a coupled calculation from an unavailable UTR
pvmcore6x_20.ii pvmcore6x.ii pvmcore6xr.ii	Semi-implicit coupling input model with 6 coupling junctions. The "6x" file runs from 0 to 10 seconds; the "6xr" file restarts at 10 seconds and runs to 20; the "6x_20" file runs from 0 to 20.

The input set comprised of r5_melcorex.ii, r5_melcoreb1.i, and r5_melcoreb2.i was useful in identifying an error in the coupling of RELAP5-3D to MELCOR involving a non-positive timestep.

Unusual.i was created to severely test the unusual timestep handler, New Algorithm 6, and it revealed some problems that were also corrected.

9.5 The Proprietary Tests

In order to provide an algorithm with the required level of robustness, a comprehensive test program was required. This test program consisted of a suite of proprietary problems that either expressed algorithmic failures in previous versions of the algorithm or problems that had been designed to test specific features in the algorithm. During the development process, new failure mechanisms in the algorithm were discovered, these proprietary problems were then added to the test suite. These problems are not available for distribution but played a fundamental role in debugging RELAP5-3D.

Every version of the algorithm that was developed was tested against the entire test suite, as it existed at the time. A checklist was developed and used for each test version. A sample checklist is included as Table 1. The checklist recorded the compilation date and time for both the RELAP5-3D and PVMEXEC executable. This information was recorded to permit reproducibility of the results, if desired.

Another unique aspect of this development effort was that every version of the files containing the shared module *idtmod* and the *dtstep* subroutine, for both RELAP5-3D and PVMEXEC programs, were automatically saved in a repository immediately following the creation of executables. This process made it possible to recreate the source code that was used for any test version of the code and provided the ability to quickly identify the changes that either solved or created problems with the algorithm.

The final version of the test suite included several smaller test suites which included:

- The DTSTEP Test Matrix (Section 10)
- The proprietary installation problems (verify)
    - A test suite of 204 different problems executing RELAP5-3D as a standalone program.
- The Weaver Failure Tests (Sec. 9.3)
- A suite of restart problems that couple RELAP5-3D and COBRA-IE (cobra_restart) This is a suite of problems that is used to make sure that coupled problems can be restarted during the solution process without altering the solution. This test suite

consists of a base case and then 5 different restart cases. Each of the restart cases starts from the previous restart case and the results are compared to ensure that results are the same to machine precision.

- A collection of 5 different User problems (User1 – User5)
  These problems represent specific problems that were uncovered in the algorithm at various points in the development process. These problems were long running jobs, with the longest taking over 4 days to reach the point of failure.

A checklist for running this collection of test suites was developed. An example is shown in Table 20 in Appendix A.

10.0 The DTSTEP Test Matrix

The DTSTEP Test Matrix was was designed as a huge collection of test cases that would exercise every set of important conditions and relevant paths through RELAP5-3D DTSTEP (R-DTSTEP) for coupled problems. Developed simultaneously with the debugging methods of Sections 8.2 through 8.4, it was designed to systematically test each mode of coupling on all known important failure mechanisms. The design is given in Sec. 9.1. The 199 debug card design is in Sec 9.2. Subroutines created to implement it are given in Sec. 9.3. The modifications to existing subroutines in reported in Sec 9.4. The large collection of scripts that run it are described in Sec. 9.5. The input models, including the newly created ones, are described in Sec 9.6. The debugging effort is reported in Sec. 9.7.

10.1 Test Matrix Design

The Test Matrix is comprised of 2856 separate test cases, each testing a combination of logic paths through the code that have generated PVM coupling errors. The test matrix is divided into five packages of test cases; each addresses a different aspect of the DTSTEP. Each contains two or more copies of the next smaller package applied differently. The packages are shown in Table 12 along with the number of separate tests in each package.

Table 14. Test Matrix levels of testing DTSTEP

Name	Description	# tests
Basic	Test special timestepping conditions (halving/doubling/repeating) and failures	17
Package A	Vary the kind of timestep size with Basic Group	34
Package B	Vary Time Target when it is applied on Group A	102
Package C	Vary type of time target: Normal, unusual, sequence, every step on Group B	408
Package D	The 7 PVM coupling modes on Group C	2856

An example would be a test that repeats the timestep (Basic) when dt is minimal (package A) at a plot pt. (package B) that is an unusual time (package C) in semi-implicit coupling (package D).

The Basic Set is comprised of simple conditions that are known to have caused failures in existing PVM-coupled input models. There are single condition tests (designated a1 through e3) and tests comprised of combinations of single conditions. These are summarized in Table 13.

Table 15. The 17 Basic Tests of the DTSTEP Test Matrix

Test Designation	Test Number	Description
a. Repeat Only (*Note: These all travel the same logic path through RELAP5-3D DTSTEP.*)		
a1.	1	succes=5 (Air appearance, velocity flip-flop, water-packing)
b. Reduction Only (halving)		
b1.	2	Exceed Material Courant Limit (MCL)
b2.	3	PVM transmits reduced step
c. Repeat with Reduction		
c1.	4	succes=1, Excessive Mass error
c2.	5	succes=2, Thermodynamic property failure
d. Amplification (doubling)		
d1.	6	MCL and mass error check
e. Code Termination for FAILURE condition		
e1.	7	fail==.true.
e2.	8	Variable volume minimum size (succes=6)
e3.	9	Executive: pvmerr /= 0
Combinations		**Conditions occurring on successive steps**
a1-b1	10	a1 then b1
a1-b2	11	a1 then b2
a1-c1	12	a1 then c1
a1-c2	13	a1 then c2
a1-b1-d1	14	a1 then b1 then d1
a1-b2-d1	15	a1 then b2 then d1
a1-c1- d1	16	a1 then c1 then d1
a1-c2- d1	17	a1 then c2 then d1

10.1.1 Package A Timestep Sizes (34 tests)

Package A applies the Basic Set twice, once with a normal sized step, and once with Δt_{small}. The two instances of the Basic set are named A1 and A2. Thus,

- A1 Basic set (9) and 8 Combinations at non-minimal time step
- A2 Basic set (9) and 8 Combinations at minimal time step

10.1.2 Package B Time Targets (102 tests)

Package B applies package A thrice, once at a minor edit, then at an explicit exchange time, and at the end of transient. The instances of Package A are named B1, B2 and B3. Thus,

- B1 Package A tested at/near minor edit time (represents plot/major/restart edits)
- B2 (where applicable) Package A tested at/near message exchange time
- B3 Package A tested at/near Transient end

10.1.3 Package C Normal/Unusual Time (408 tests)

Package C applies package B four times, once at multiple of DTMAX, then at an unusual time, at a sequence of normal and unusual targets, and at every timestep. The instances of Package B are named C1, C2, C3 and C4. Thus,

- C1 Package B: Time targets are integer multiples of DTMAX
- C2 Package B: Time targets are unusual (non-integer multiples of DTMAX)
- C3 Package B: normal target, followed by unusual, followed by normal
- C4 Package B: Targets specified at every step via RELAP5-3D or Executive.

10.1.4 Package D Coupling Configurations (2856 tests)

Package D applies package C seven times, as indicated in the bullets. The instances of Package C are numbered D1, through D7. Thus,

- D1 Stand alone RELAP5-3D
- D2 Explicit Asynchronous coupling RELAP5-3D/Relap5-3D
- D3 Semi-implicit Synchronous coupling RELAP5-3D/Relap5-3D
- D4 Simultaneous coupling (Explicit Asynchronous and Semi-implicit Synchronous) RELAP5-3D/Relap5-3D/RELAP5-3D
- D5 Explicit Synchronous coupling RELAP5-3D/Relap5-3D
- D6 Explicit Asynchronous coupling RELAP5-3D/Relap5-3D with 1st as leader
- D7 Synchronous single RELAP5-3D controlled by Executive

Note that there are somewhat fewer tests than the upper limit of 2856 tests in the matrix. For instance, asynchronous coupling implies no testing of Basic Test b2 and that there can be no testing of sub-package B2 with semi-implicit coupling, D4. There are other combinations that make no sense to test as well. The run scripts prevent non-sensical cases from being run; see Sec. 9.4.

The designation of a single case within the DTSTEP Test Matrix is unique. It is named according to its groupings, from largest package to smallest. The format of every case is:

Dv Cw Bx Ay CASE z

An example is D2 C4 B3 A2 CASE 10 indicates that the case tests asynchronous coupling with minor edits output every time step wherein air-appearance followed by a violation of the material Courant limit occurs just at the end of the transient. The subroutines described in Sec. 9.3-9.5 use no spaces in the naming the cases. E.G. DvCwBxAyCASEz

The test matrix is designed to take a single input model, one from each coupling configuration in Package D, and modify it to apply the specifications of the other packages to the input deck to create an altered deck that exactly defines the test to RELAP5-3D. Modifications to the input deck are made by scripts and include the addition of the new 199 debug card; the subroutines convert the input into logic flowpath control.

10.2 The 199 debug card

Part of the Test Matrix design is a new input card, number 199, contains information about which subroutine to test, what to test, and when to test it. However, the 199 card is designed to be a generic system for creating user-controlled debug information for use ANYWHERE in RELAP5-3D.

The Generic 199 card is a standard RELAP5-3D input card. Its format is:

199 Word1 Word2 Word3 Word 4

- Word 1 A character string that names a subroutine (group) to debug
- Word 2 A keyword that indicates what aspect to debug
- Word 3 A number indicating the advancement to activate the debugging
- Word 4 A number indicating the advancement to terminate the debugging

The 199 card is limited to DTSTEP debugging currently. Therefore, Word 1 must be DTSTEP and Word 2 must be a basic-case-number, z, followed by package A subset indicator, either A1 or A2. Thus the 199 card Word 2 must have the form zAy. The 199 format is:

199 DTSTEP "zAy" T1 T2

where $1 \le z \le 17$, $1 \le y \le 2$, and the test activates on advancement T1 and terminates on T2. By choosing T1 and T2 properly, the time-target aspect controlled by package B is incorporated into the test case.

Thus, the 199 card creates the aspects of the test case controlled by packages B, A, and Basic, but not packages C and D; those are controlled by the scripts. Exercising the specified Test Matrix case is left to the new Test Matrix subroutines.

10.3 The new Test Matrix subroutines

Two subroutines, with 4 internal subroutines, were created to implement the DTSTEP Test Matrix; they are described in Table 14. Module Testmod carries the data created by the input one to the transient routine. Testmod is described in Sec 6.4.4 in Tables 6 and 6a.

Table 16. Subroutines created to implement the DTSTEP Test Matrix

Major Subroutine	**Description**
rdebug	Read user-control debug data from 199 card, check it and print diagnostics for for incorrect form or data. Interpret data as Table 13 “Test Designation” (with no dashes) in Testmod variable ctest.
testDtstep	Create tests of various logic paths through DTSTEP for the DTSTEP Test Matrix by interpreting ctest and setting variables cpurei(4:5), done, errmax, fail, success, and/or iTestDtstep appropriately on the user-designated timesteps only.
Internal Subroutine	**Within testDtstep**
basicSet	Sets DTSTEP variables to implement test from Table 13 “Test Designation”
combine1_4	Sets DTSTEP variables to implement test of Table 13 first 4 combinations.
combine5_8	Sets DTSTEP variables to implement test of Table 13 2nd 4 combinations.
packageA_set	Conditionally calls subroutines basicSet, combine1_4, or combine5_8.

The new subroutines are internally documented. Except for a few local variables, every variable used by them is listed in the preceding Tables of variables.

10.4 Updates in support of the Test Matrix

Subroutine RNEWP was adjusted to call RDEBUG. Two RELAP5-3D subroutines TRAN and DTSTEP were modified to implement the Test Matrix in the transient. Both access the data in Testmod to activate the tests. Three more subroutines were modified after version r3d244b to identify lapses in writing major/minor/restart edits in Test Matrix output files automatically. A protocol to reduce PVM daemon acquisition errors was installed in PVMEXEC TRAN and RELAP5 TRNCTL.

In RELAP5-3D TRAN, the Test Matrix is activated by a call to testDtstep just above the coding for the first Weaver Test, marked testpvm1; see Sec. 8.3 above. It is called again just prior to the call to DTSTEP. Both calls occur in coding marked with pre-compiler directive pvmcoupl so that it can be removed for most client groups.

In RELAP5-3D DTSTEP, the coding is implemented with if-tests on variable iTestDtstep. All these if-tests are also marked with pre-compiler directive pvmcoupl. The implementation in DTSTEP is described in Table 15.

Table 17. Implementation of Test Matrix basic tests within RELAP5 DTSTEP

iTestDtstep	Description	Implementation in RELAP5-3D DTSTEP
1	Exceed Courant limit	Forced by setting dtx to the larger of Δt_{small} and $0.9x\Delta t_{i+1}$ in Sec. 10.3 of DTSTEP
2	Halving dt	Forced by resetting dt=dt/2 in DTSTEP Sec 11.1 (synchronous coupling only) just before the proposed dt is sent to PVMEXEC. This is necessary so that PVMEXEC has same dt.
3	PVM error condition	Forced by setting pvmerr=1 in four places: in initialization, within IntTarg8000, in Sections 2.3 and 7.1 of DTSTEP. This is necessary to avoid creating a machine hang.
4	Minimal step size	Forced by setting dtx to twice dtmin in Sec 10.3 of DTSTEP

In order to verify that the major, minor, and restart edits are occurring as they should, coding was added to count the number of each, and to output those values with each message about the appropriate edits in the “printed output” file. These changes occur in MAJOUT, MIREC, and RSTREC. Even then, the scripts automate the process of checking the values for correctness.

A common problem related to running numerous successive PVM jobs was failure to obtain a PVM daemon to start running a job. This would occur when the Test Matrix was run. To overcome this, an optional “permission to terminate” (also called “Mother may I”) protocol with PVM was developed. The message tag is 8100.

Just before the return statement in TRNCTL of RELAP5-3D, an 8100 message is sent to the Executive. This is followed by a PVM-Receive statement. The process cannot terminate until either an 8100 message from PVMEXEC TRAN arrives, or the wait time is exceeded. The executive has a receive loop similar to an 8005 message tag in TRAN and only after all processes have requested termination may the executive broadcast the terminate message to all. If one of the child processes fails, the PVM Executive wait time will be exceeded and the normal error message will be written from the main program.

10.5 The New Scripts

A large set of Linux scripts was created to organize, set-up, run, and check the DTSTEP Test Matrix. These are summarized in Table 16.

Table 18. The new scripts that operate the DTSTEP Test Matrix

Script Name	Function	Calls	Called by
base17	Run 17 basic test cases of DTSTEP in sequence. Its 4 call arguments describe the choice of D-coupling, C-kind, B-target, and A-dt-size	mk199card settimecards testsuccess	packageA
BasicCase	The centerpiece of the scripting system. Given the 5 numbers that describe a test, it selects the input model template and prepares the executive input file, makes copies of the slave process input files, adding a debug card, and modifying their time cards. It also runs the code, examines the output and determines if it is satisfactory before cleaning up and renaming the output for later manual examination. BasicCase can be run manually for any of the test matrix's 2856 cases.	checkCase mk199card settimecards testsuccess pvmexec.x relap5.x	base17
CheckCase	Determine if a case makes sense to run. Return with status flag 1 if the Basic Case makes no sense to run. Also display a message explaining why.	none	BasicCase
editCount	Counts the number of restart, major, and minor edits in the RELAP5-3D output file.	None	BasicCase
mk199card	Creates the 199 debug card to append to the end of a RELAP5-3D input deck that implements the test specified by packages B, A and BasicCase (time, dt-size and basic case number)		BasicCase
packageD	Run all 2856 problems of the DTSTEP Test Matrix by calling packageC for all 7 types of coupling tests. Stores the results in a log file.	packageC	runTest
packageC	Runs 408 test cases by calling packageB for 4 kinds of time targets (normal, unusual, combo, every-step).	packageB	packageD
packageB	Runs 102 test cases by calling packageA for 3 target types (plot, exchange, transient-end)	packageA	packageC
packageA	Run the 34 basic test cases for the conditions of A1 and A2 for a normal and smallest time step respectively.	base17	packageB
rerun	Finds and reruns all Test Matrix cases that ran and produced the result "Unsatisfactory."	Creates reRunTest	none
runTest	Set up the DTSTEP Test Matrix, making certain that the RELAP5-3D, PVMEXEC, and pvmcatchout executables and all the fluid property files are accessible. Also deletes all output files from any previous Test Matrix run in the directory.	packageD	../runT
../runT	Runs the DTSTEP Test Matrix and DTSTEP Extra problems. Checks first that the code was installed with PVM on. The log file defaults to the date followed by _TM.	runTest (Test Matrix) runv_dt (of Sec. 8.4)	None
seek	Checks that a single success value occurred in at least once in a group of files.	None	testsuccess

settimecards	Creates timecards that implement the test called for by packages B, C, and D.	None	BasicCase
testsuccess	Compares the output of a run of the PVM-Executive (or RELAP5-3D in the case of set D1) against the expected final message.	seek	BasicCase

All scripts in Table 16 are located in the TestMatrixDt subdirectory of the ”run” directory, except runT which is located in “run.” More complete testing is available by running runT from the “run” directory of RELAP5-3D; this will run all test cases listed in Sec. 8.4 and the Test Matrix.

As described in an earlier Section, the 199 card does not communicate the D- and C-level aspects of a Test Matrix test case. These are introduced by two mechanisms. Script settimecards of Table 15 modifies the timecards of an input deck to implement the C-level control. BasicCase selects the input deck that corresponds to the D-level coupling of the specific test case.

10.6 The Input Models

Simple input models are used as the basis for all tests of a particular kind of D-level coupling. The input models are given in Table 17.

Table 199. Test Matrix base input tests, the models and their sets of input files

D-level	Base Input File	Other input files of base set	Description
1	edhtrk.i		RELAP5-3D standalone. Edward’s pipe model (with extras). Reference [c].
2	pvmedax.ii	pvmedac.i – child pvmedap.i – parent	Asynchronous explicit coupling test case. Edward's pipe
3	pvmcore	pvmcorep.i – primary pvmcorec.i – core	Semi-implicit (synchronous) coupling Christensen model
4	pvm3way	pvm3wayp.i – primary pvm3wayc.i – core pvm3wayb.i – bypass	Semi-implicit (primary to core) AND explicit asynchronous (primary to bypass) coupling Christensen model. Reference [d].
5	pvmedspx.ii	pvmedsc.i – child pvmedsp.i – parent	Synchronous explicit coupling Edward's pipe
6	pvmeds10x.ii	pvmeds10f.i – follower pvmeds10l.i – leader	Asynchronous explicit conserving Edward's pipe
7	pvmvesselx.i	pvmvesselp.i	Executive oversees single RELAP5-3D running a full Christensen vessel model

There are only two basic input models, one based on the Edward’s blowdown test, and the other based on the Christensen vessel model. These problems are simple and fast running as compared with, for example, a full plant model or the standard Typical PWR model that has serious modeling deficiencies that cause many code failures. There are two reasons for this.

The first reason is time; there are 2856 test cases and the time to run the full Test Matrix, at the time, was 2.5 to 4 hours, depending on the computer platform. Changing from Edward’s pipe that runs in a few seconds to a problem that runs in minutes or longer would make the Test Matrix too slow for debugging purposes. Increasing the runtime of the base input models by

using something that took a minute to run would increase runtime by a factor of 20 or more and increase Test Matrix runtime from 4 hours to 4 days. This was unacceptable as the test matrix was run hundreds of times during the course of testing and debugging.

The second reason is failures. The PVM system would fail by itself for various reasons including inability to get a PVM daemon, required failures (basic cases 7, 8, and 9), and DTSTEP bugs. Simple, stable models were needed to identify these errors. Errors in other parts of the code that may or may not have originated from coding bugs in one of the DTSTEP programs would only complicate and further lengthen the overall debugging process.

10.6.1 The New Input Model

To perform coupling tests in the D4 category, a new simple test problem was required. This was made by decomposing the "pvmcore" problem so that a portion of the core was connected to the primary part of the vessel (including the rest of the core) semi-implicitly, while the bypass was connected to the primary explicitly.

The new problem is called "pvm3way" and a nodding diagram is given in Figure 5.

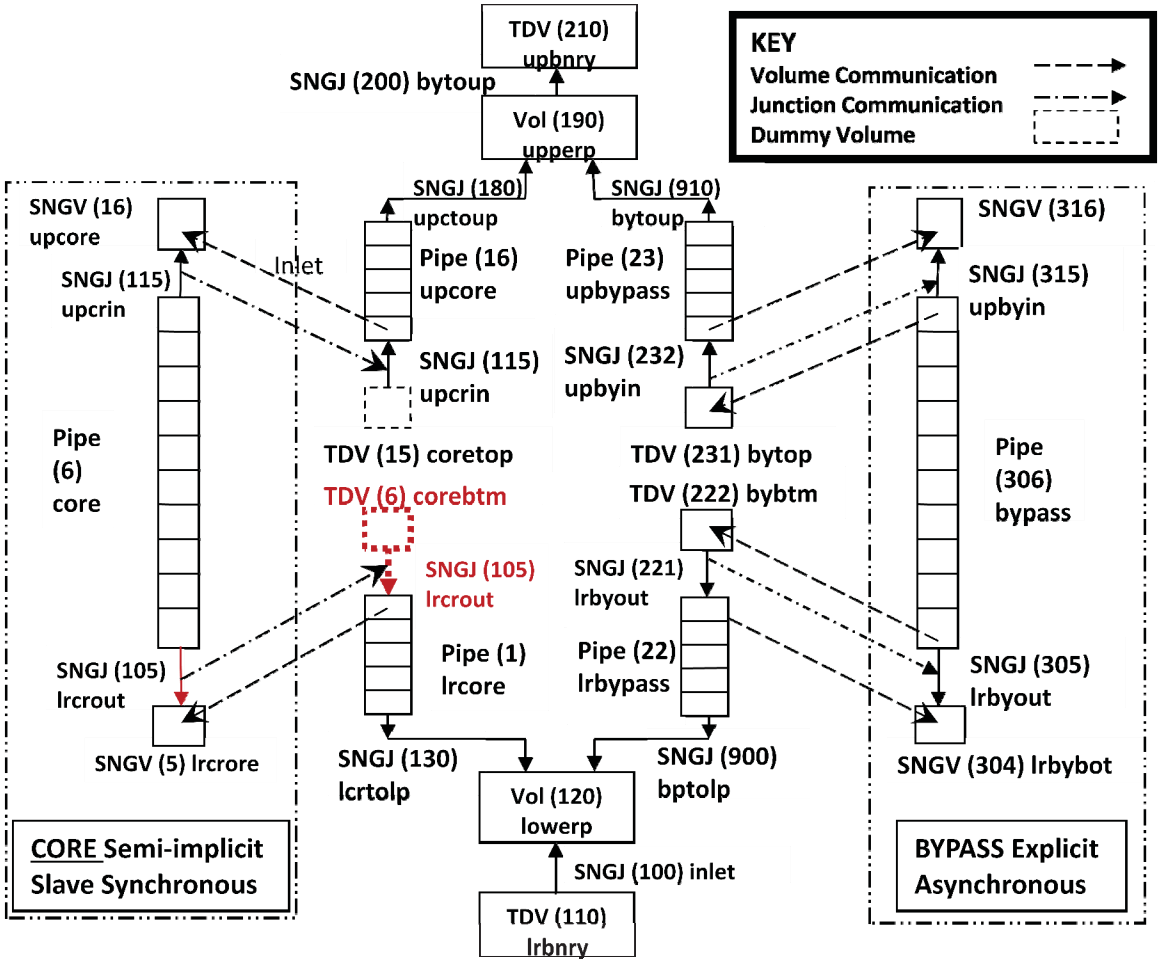


Figure 5. PVM3WAY Nodalization Diagram.

11.0 Supporting Changes in Existing Coding

This Section provides a brief summary of the existing subroutines in both RELKAP5-3D and PVMEXEC that were modified to implement the integer timestep algorithm and in support of debugging the coding.

Subroutine pvmfxrec within RELAP5-3D was changed to activate the debug print statements based on the variable from the shared module. The debug print statements now are sent to a common file unit. The unit number is stored in *idtmod*.

Subroutine trnctl of RELAP5-3D was modified to incorporate the "Permission-to-Terminate" logic for that mitigates PVM failures due to the Operating System shutting down buffers of terminating processes before its data exits to coupled receiving process (Sec. 10.3 and 11.4).

Subroutines majout, miedit, and rstrec were modified to implement the DTSTEP Test Matrix (Sec. 11). Subroutine majout was also modified to display in excess of 14 digits and/or display in hexadecimal for some important debugging quantities.

Module contrl added a new variable, old_pvm_samedt, has been added for the detection of potential infinite loops (Section 5.6 and 12.0).

Subroutine tran of PVMEXEC was modified to incorporate logic for the detection and mitigation of potential infinite loops (Section 5.6). This is done by saving a new variable, old_pvm_samedt, which saves the state of the variable pvm_samedt from the previous timestep. If both pvm_samedt and old_pvm_pvm_samedt are non-zero, the code enforces a timestep size reduction.

Subroutine inputd of PVMEXEC has been modified to provide the ability to provide debug print statements.

12.0 PVM-coupling and DTSTEP Problems solved

The following are a list of some of the problems fixed during the development and debugging of the PVM coupling issues with DTSTEP. In this section, R-DTSTEP refers to RELAP5-3D DTSTEP and P-DTSTEP refers to PVMEXEC DTSTEP.

Errors reported here are confined to DTSTEP errors that existed prior to the implement of the integer timestepping algorithm and its development. Many other errors were corrected along the way including errors in the integer timestepping implementation itself, errors in the implementation of the DTSTEP Test Matrix, and errors in other subroutines than DTSTEP.

PVM0901 Code hangs at an unusual endtime Floating point error
Description: Hang with floating point time control at unusual times sometimes.
Analysis: The user-supplied problem, time8.i, illustrates several types of code failures when RELAP5-3D is run under the control of the PVMEXEC without being coupled to anything.

Timecard 6 ends at 25200.0021 s with DTMAX = 0.004; thus the endtime is unusual and was illustrative of code hangs experienced prior to the integer timestepping project.
Solution: This was resolved by converting to integer timestepping.

PVM0902 Modification for Explosive Doubling PVM conceptual design error
Description: During the final timecard, the timestep begins to double, exceeds DTMAX, and continues doubling until the end time is reached.
Analysis: This is a problem in explicit asynchronous coupling only, since for any synchronous coupling the RELAP5-3D timecards have no control and for RELAP5-3D standalone, there are no overriding timecards. An error with synchronous processing arose because the method implemented to overcome explosive doubling affected the synchronous case.
Solution: Adding a clause to prevent synchronous processes from entering the explosive doubling control coding (reported in February 2009) resolved the problem.

PVM0903 Cumulative time in message 8000 PVM conceptual design problem
Description: The code hangs at the end of timecard; see Section 5.5.
Solution: The cumulative time was added as a ninth data item to both P- and R-DTSTEP. This allows all slave processes to stay in lock-step with the PVM Executive.

PVM0904 Penultimate timecard stop Old DTSTEP conceptual design error UP 03024
Description: When running a D4 Test Matrix case, it was discovered that some asynchronous processes would stop at the end of the second to last timecard.
Analysis and solution: This was tracked down to an error in resetting variable curclm in Section 7.1.1 of R-DTSTEP. This error has been fixed.

PVM0905 Hang when HYDRO sets FAIL PVM conceptual design error
With D5, code hangs for case D5C2B1A1 case 7. Reported on 24-Mar-09.
Analysis: The algorithm designer did not realize that HYDRO could set FAIL to TRUE. When this happens, RELAP5-TRAN send the (A-OK) 10000 message to the Executive, but later during the time-step DTSTEP correctly begins the shutdown process. Thus both codes arrive at different message wait locations.
Solution: When fail is true from hydro, in the immediately-following PVM coding section, set PVMSUCCESS = 5 to signal a repeat (because RELAP5 will repeat, then shut down).

PVM0906 Hang in explicit asynch. sequential PVM conceptual design error
The code hangs for case D6C1B1A1 cases 10-12. Reported on 25-Mar-09.
Analysis: When the leader of a sequentially coupled problem reaches the end of the interval, it sends a message to the follower to proceed. Normally the follower receives the message in RELAP-TRAN and proceeds through advancements to the rendezvous time. However, if on the same advancement as the receipt of the go-ahead message is received, the code must repeat the advancement, it can repeat the arrival at the receive message. This occurs on, for example, D6C1B1A1 case 10.
Solution: The solution was to create a logical variable that DTSTEP sets whenever the code must repeat the time step. RELAP-TRAN was then modified to access the section of code containing the above-mentioned receive message only when that variable is FALSE. A temporary module,

BACKMOD, was created to carry it. This was done to make it more obvious when porting to F90 where the variable will be absorbed into a more appropriate module.

PVM0907 Code hang on D4C2B1A1 case 1**.** PVM design error
Description: Hang with D4C2B1A1 case 1 by stretching past explicit exchange time
Analysis: The value of CURTEX was not set to the endtime of the timecard, an unusual time, but rather the multiple of DTMAX just before it. It did not perform the stretch calculation.
Solution: The calculation of integer time targets was modified to incorporate the stretch logic. Thus if I_CURTEX + 10%IDTMAX > IDURATION (relative integer time end-of-timecard), then reset I_CURTEX to the nearest multiplier of DTSMALLEST to the end time of the timecard. Then CURTEX is set to the endtime if I_CURTEX is close to the integer timecard end time, IDURATION.

PVM0908 D4C4B1A2 cases 12 and 16 hung Old DTSTEP error
Description: Hang from repeat condition at minimum dt (expl asyn parallel)
Analysis: When SUCCES == 5 the code errs at minimal time-steps. In R-DTSTEP Sec. 2.3.3, the code halves DT when SUCCES /= 5 despite DT = DT_MIN. It then doubles the time-step in Section 2.4, which can only be reached when SUCCES /= 1. When DT == DT_MIN and SUCCES == 5, the code therefore sets DT = 2*DT_MIN. This is incorrect and causes the slave process to take twice as big a time-step as the executive takes. As a result, they arrive at different messages to send and listen.
Solution: Change the test in DTSTEP Sec 2.3.3 so that both SUCCES /= 5 and DT /= DT_MIN, in order to halve DT. Remove the doubling of DT in DTSTEP Section 2.4. All other paths through the code work the same because the halving and doubling cancel. This change actually makes the code more efficient by reducing wasteful operations.

PVM0909 PVM-dtstep Dead Coding PVM design Issue
Analysis: P-DTSTEP was taken from R-DTSTEP and reduced. Some sections of the remaining code are no necessary and can be removed, simplified, and/or corrected. In version 2.4.1.2 P-DTSTEP, the statement “if (dt > dt_min(i))go to 70” statement is ***always*** taken; so the coding thereafter to statement 70 can never be accessed and was removed. Moreover, in R-DTSTEP, dt is halved if SUCCES is non-zero, and this test divides between minimal and non-minimal time-step handling. However, halving DT is not done in P-DTSTEP and so DT is always greater than the user input minimum at that point in the calculation.
Solution: The coding beneath it was corrected to write proper error messages, not adjust the time or time-step, not reset PVM_SUCCES, and jump to the 8001-message section of coding.

PVM0910 Stretch Logic Error Corrected PVM implementation error
Analysis: For a couple synchronously coupled problems, R-DTSTEP recognizes a stretch unusual time, calculates half the distance to the unusual time as time step, takes it, but on the next advancement, takes a shorter time step that reaches the next multiple of DTMAX. It thus ignored the stretch logic on the second step of the stretch handler logic. The Executive and child take different time steps and the coupled process hangs.
Solution: in R-DTSTEP

- In dthyCalc: DTHY -> DTMAX_I
- In Unusual Handler: EDITTIME -> CURTMIN = min (endInterval(i), curtxy) where xy ε {mi, mj, rs, ex}.

Solution: in P-DTSTEP

- Reset all curtxy if |endtime – curtxy| < dtsmallest

PVM0911 PVMEXEC Input Error Messages PVM implementation deficiency
Analysis: Compared to RELAP5-3D, PVMEXEC provides little information to users about input errors. In fact under some circumstances, it will fail input with no message at all, such as if a blank line occurs after the executive's final timecard.
Solution (partial): A dozen new error messages were added to PVMEXEC in the subroutines INPUTD and RNEWP.

PVM0912 64-bit error in version 2.4.4 Script and source code errors
All PVM installation test cases failed in 64-bit installation mode.
Analysis: PVM Executive declared WAIT to be real*4, but describing it as real*8 in the PVM message because of an error in dpvmexec, the installation script. Also, ETIME was not declared REAL*4 in TIMSET in the envrl folder.
Solution: The script was corrected and the variable was declared properly.

PVM1001 Coupled Code Induced Infinite Loop New Coupling-only Error
Synchronously coupled codes enter an infinite loop caused by one code forcing a timestep repeat which causes the other to force timestep repeat in an unending cycle; see Figure 6.

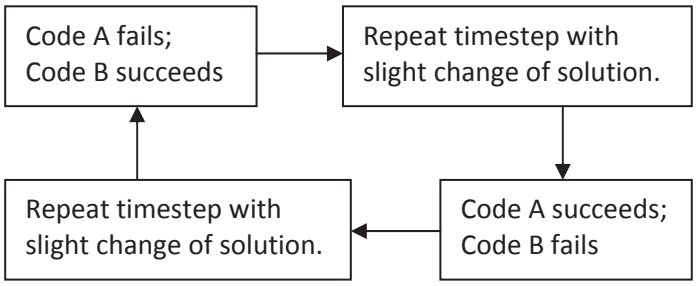


Figure 6. PVM3WAY Nodalization Diagram.

Analysis: This can occur when two or more codes each detect a failed timestep size that does not require a reduction in timestep size to clear. For RELAP5-3D, air appearance and velocity flip-flop are two mechanism that cause timestep repeat with same timestep size and slightly different conditions. The slightly different conditions then cause another coupled code to fail the advancement without requiring a timestep size reduction. If this failure results in a return to the conditions that caused the first code to fail, an infinite loop occurs; see Figure 6. Neither code can detect this condition itself.
Solution: Since the coupled analysis codes cannot detect this condition, code was implemented in PVMEXEC to detect the scenario. If two-successive timesteps have been failed at the same timestep size, PVMEXEC changes the timestep status from "pvm_samedt" to "pvm_smallerdt." Further, the original algorithm, which never ensured that a smaller timestep size was used if a status of "pvm_smallerdt" was received, was upgraded so whenever the "pvm_smallerdt" status is received from any of the codes, the largest timestep size that is allowed on the repeated timestep is ½ of the previous timestep size.

13.0 Conclusions

A new integer-based timestep selection and advancement algorithm has been implemented in the PVMEXEC and RELAP5-3D computer programs. A comprehensive code test suite was developed. The algorithm currently has no known problems associated with it. The work to develop the new algorithm was completed in time to support the specific need dates for a robust algorithm.

14.0 Acknowledgements

The work on this project was supported by consultants Dr. Walter Weaver and Hope Forsmann. Walt was a seminal author of the PVM coupling and provided explanations of many subtleties of the workings of the coupling as well as detailed insight into the coding. Hope was steadfast and instrumental in tracking down several of the errors listed in Section 11.

Several staff members at funding source were involved with the debugging and solving of user problems, creation of solutions to those problems, and their implementation of those solutions in the source code.

15.0 References

(a) Weaver, W. L., Tomlinson, E. T., Aumiller, D. L., 2002, "A PVM Executive Program for Use with RELAP5-3D," Proceedings of ICONE-10, Arlington VA, April 2002.

(b) W. L. Weaver, "Programmers Manual for the PVM Coupling Interface in the RELAP5-3D© Code," INL/EXT-05-00203, March 2005.

(c) A. R. Edwards and F. P. O'Brien, "Studies of Phenomena Connected with the Depressurization of Water Reactors," Journal of the British Nuclear Energy Society, Vol. 9, 1970, pp. 125-135.

(d) H. Christensen, "Power–to–Void Transfer Function," ANL-6385, 1961.

Appendix A

Table 20 – Sample Testing Checklist

RELAP5-3D data

Location of executable	/mount/betp_p18/relap5.x
Compile Date	20100514
Compile Time	12:32:04

PVMEXEC data

Location of executable	/mount/betp_p18/pvmexec.x
Compile Date	20100512
Compile Time	09:41:24

Testing Status

Problem Name	Status	Comments
Matrices	Works	
verify	Works	
misc_cpl	Works	
cobra_restart (enhanced)	Works	
R5_melcor_hstr	Works	
long_running_single_R5	Works	
User1	Works	
User2	Works	
User3	Works	
User4	Works	
User5	Works	
R5_melcor – ss_complete	Works	
R5_melcor – tran1	Works	